



PrimeFactors™
— APPLIED DATA PROTECTION —

WHITEPAPER

What Enterprises Need from Tokenization

February 2020

Introduction

We live in the digital era, where more data is being generated by more applications than at any point in history and the value of data has never been higher. Business leaders continue to deal with the aggressive pace at which computer systems have become interconnected, distributed, and cloud deployed. Market pressures have made offering always-on, available-everywhere access to data and applications a baseline requirement, with the unfortunate side effect that remote, “faceless” electronic thefts of sensitive data through various types of data breaches have become commonplace and continue to torment companies and governments around the globe.

Nearly two-thirds of global companies have reported some form of data security breach at some point in the past, and in the last year alone, nearly half of American companies have reported a data breach¹. The analyst firm, Gartner, reports that most of the successful data breaches happen at the application layer, where data is more exposed and susceptible to breach, and regardless of their methods, most attacks are targeted on obtaining data². Billions of dollars are spent each year on various types of cyber security, yet intrusions continue to persist, and sensitive data remains at risk. A common strategy for minimizing the amount of rework required to introduce appropriate data protection into existing applications and data sources is to substitute surrogate data elements in place of *Personally Identifying Information* (PII) and other sensitive data — that is, to replace things like credit card numbers, *Social Security numbers* (SSNs), healthcare data, and other private information with surrogate values, or *tokens*. This tokenization of data at the application level provides multiple benefits and can be a simple and versatile solution for safeguarding data where it is most vulnerable.

Tokenization Defined

Tokenization, in the context of electronic data protection, is the automated process of substituting a surrogate value (token) for a sensitive data value in a processing system. These surrogate values could be *reversible tokens*, which are able to be returned to their original data value, or *irreversible tokens*, which are permanent and cannot be reidentified. When using reversible tokens, effective commercial implementations must allow for the high-speed cross-reference of the surrogate token to the original data, when the original value is required. However, such cross-reference should be limited to authorized users who can access the original sensitive data and must be otherwise impervious to penetration by cyber thieves and other unauthorized users.

Recognizing the value of sensitive data and the harm that could be caused if certain data were to fall into the hands of the wrong parties, many governments and industries have established laws and compliance standards by which sensitive data must be protected. Data protection regulations that specifically acknowledge the need to protect the personal information rights of citizens have been enacted by many governments around the globe, including most state governments in the United States of America. These ‘personal privacy’ regulations often contain broad sets of rules that govern how organizations capture, control, process and protect personal information, placing the responsibility for the protection of personally identifying information directly on the organization that collects it. The European Union’s *General Data Protection Regulation* (GDPR)³ was one of the first of these types of

regulations that opened the door to catastrophic fines levied against entities that fail to comply with standards for privacy protection. The US federal *Gramm-Leach-Bliley Act* (GLBA) ⁴, the *California Consumer Privacy Act* (CCPA) ⁵ and Brazil's *General Data Protection Law* (LGPD)⁶ are additional examples of this global movement to regulate the protection of privacy. Tokenization can be used to meet two specific requirements contemplated in some of these data privacy regulations – *pseudonymizing* and *anonymizing* sensitive data.

Pseudonymization

The most common approach for pseudonymization is tokenization. Pseudonymization, as in reversible tokens, takes identifying data and replaces it with a value that cannot be linked to a specific individual without additional information that can be accessed elsewhere. A good comparison from the literary world would be to read a collection of articles written by Mrs. Silence Dogood. You might easily observe that each article in the collection was written by the same person, but with the appropriate insider information, you could find out that the writer's name was actually a pseudonym for Benjamin Franklin.

Anonymization

Anonymization, as in irreversible tokens, replaces original clear data with a value that is both unrelatable to the original data and permanently irretrievable. Anonymized data can never be re-associated with their original data source. Continuing with the literary example above, consider Letters to the Editor authored by "Anonymous". For a collection of such pieces, you would have no way of knowing or finding out whether they were all written by the same person or many different people. Anonymization is most often used when the original source of data never needs to be or is not allowed to be disclosed, such as in the case of a medical study.

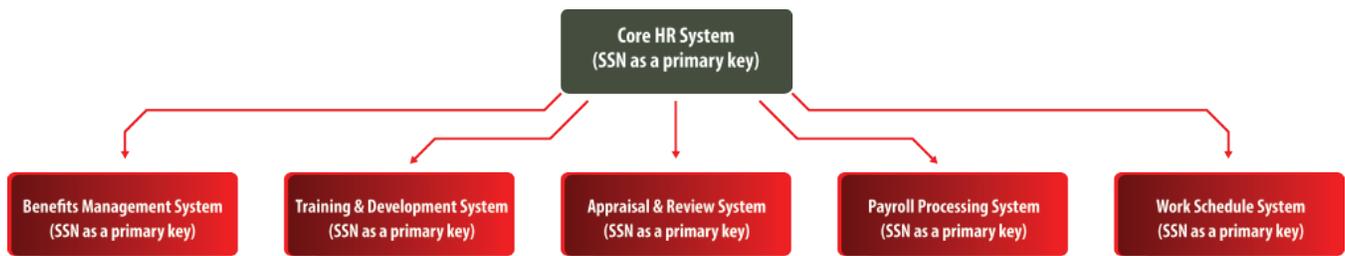
Tokenization and the Enterprise Today

While many industries set standards for pseudonymizing and anonymizing sensitive data, the *Payment Card Industry* (PCI) is perhaps the best example of industry-driven requirements for tokenization. Many business leaders adopt tokenization to reduce the "attack surface" – the number of places where sensitive data is being used, moved, or stored – and where they are therefore most susceptible to breach. For example, payment card numbers (credit debit cards) are very attractive targets for cyber thieves, particularly when they can be obtained in large volume. Systems that maintain large arrays of payment card transaction details become very attractive targets for the thieves, and the remote access, "faceless" nature of attacks on such systems lowers their risk of being identified and captured.

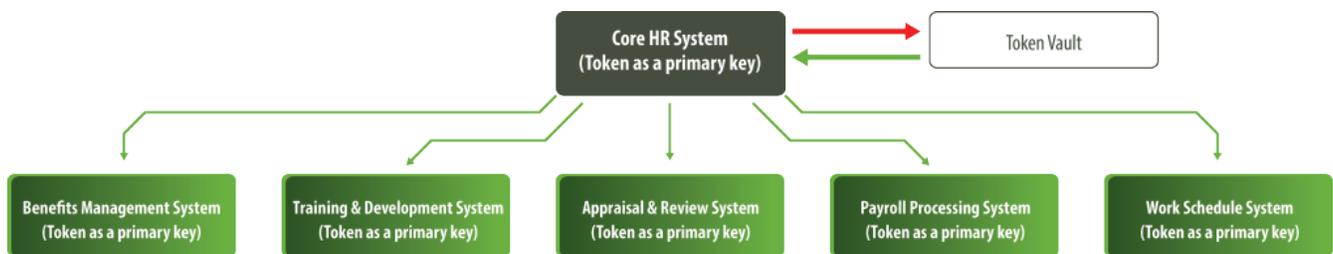
PCI leverages tokenization to replace payment card numbers when used in an enterprise payment application. Systems that do not require an actual payment card number for their processing are provided the surrogate token instead. These surrogate tokens are used as though they were the original card number in the various systems, applications, or eCommerce sites that need to reference the payment information to facilitate a cardholder transaction, however the tokens themselves would be meaningless and provide no value if they were obtained in a data breach.

Though tokens may be used broadly across an enterprise, the actual sensitive data is typically centrally held in an isolated and easier to defend token vault – only here can a token value be cross-referenced to its corresponding sensitive data. The attack surface is thereby greatly reduced, and the risk of data breach is proportionally diminished.

Another great example of leveraging tokenization to reduce an attack surface related to PII can be observed in the hiring of a new employee, whose SSN is stored in the company’s Human Resource (HR) System as the employee’s prime identifier. The SSN may also need to be shared with the various systems related to this employee’s compensation, benefits, and performance. Since each of these systems holds the PII independently, they each represent a point (or area) where unauthorized users might attempt to access this potentially valuable identity data.



In deploying a tokenization solution, the company replaces the SSN with a surrogate token and store the encrypted SSN in a secure token vault. The HR system and each of the related systems then leverage the token value, not the SSN, as the employee’s primary identifier. If any of these systems were breached, only the tokens would be compromised, not the actual SSNs, and these tokens would serve no commercial value for identity theft.



Equally, or perhaps even more importantly, tokenization can reduce the costs associated with the mandatory technology audits imposed by the payment card industry. When it is documented that systems only have access to a token, not sensitive data, they can be excluded from a PCI-DSS audit, greatly reducing the time – and consequently the cost – required to complete the necessary audit inspections. For example, if you were to encrypt an account number on a client computer with an encryption key (either in the computer memory or in a hardware encryption device), you would be required to perform an audit for how the cryptographic key was entered and managed. However, when tokenization is used, key management moves from the client computer to a centralized token server. Additionally, random tokens do not use encryption keys and therefore do not require an encryption key

management audit under PCI rules. *Removing localized cryptographic keys from PCI data protection, or in the case of random tokens – removing keys altogether, inherently limits the scope of a PCI audit.*

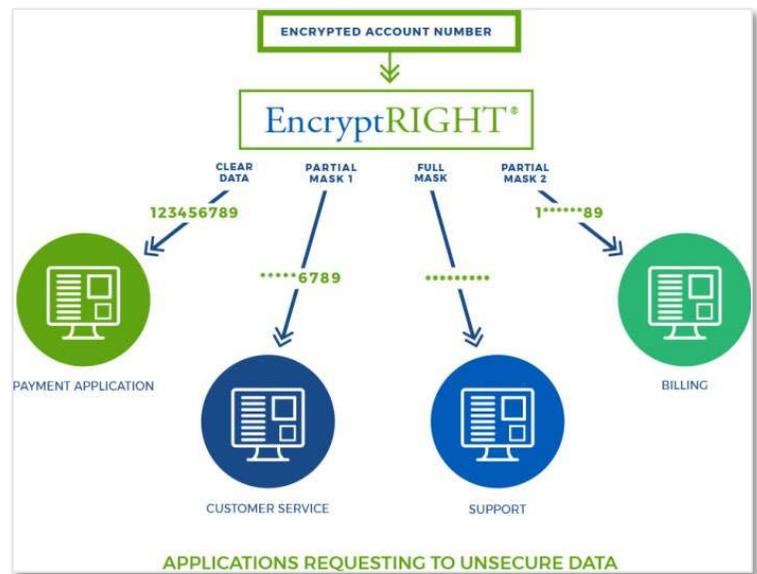
The use of tokenization to reduce potential audit scopes, comply with regulations, reduce contingent liability associated with potential data breaches, and increase tangible effects of data protection altogether, provides value in many industry use cases. Securing *Protected Health Information* (PHI) is required in the US by aspects of the Health Insurance Portability and Accountability Act (HIPAA)⁷ and the subsequent *Health Information Technology for Economic and Clinical Health Act* (HITECH Act)⁸. Surrogate token values could be substituted for values indicating the clinical results of blood tests or biopsies in billing or insurance systems where that detail is no longer appropriate. Most clinical trials use irreversible tokens, which cannot be tied back to personally identifying information, to meet anonymization requirements. Banks seeking to comply with consumer banking privacy provisions of the Gramm-Leach-Bliley Act, government agencies managing details of juvenile criminal records, Social Security numbers, or voter registration records, and any enterprise seeking to diligently, cost-effectively, increase protection of sensitive information currently processed by their systems, with minimum reprogramming and retraining – all can apply tokenization to achieve their goals.

Tokenization, Data Masking, and Managing Data Privacy

Often, different people in an organization require different levels of access to data based upon their role. For example, an HR supervisor may require access to data in the clear, while a service center representative only needs to see the last four characters of an SSN to verify an employee's identity and someone in payroll only needs to access a token to process a reimbursement. Protecting data to meet these individual needs might involve tokenization, encryption, or data masking, concerning which, there is much industry debate regarding how to strictly define the boundary between each.

In brief, **data masking** is a general method of obfuscating some or all of an authentic piece of data in a manner that protects the actual data from being fully viewed, and various encryption or tokenization techniques may be employed to establish a data mask. Data masks can be *full* (concealing all of the original data characters) or *partial* (obscuring only some of the data characters). Tokenized or encrypted data can generally be described as masked data and data that is partially masked can be referred to as partially tokenized/encrypted data, with some accuracy.

In any case, accessing data in different formats – clear, partially masked, or fully obscured – based upon the role of the user accessing the data is important in helping companies manage their various data

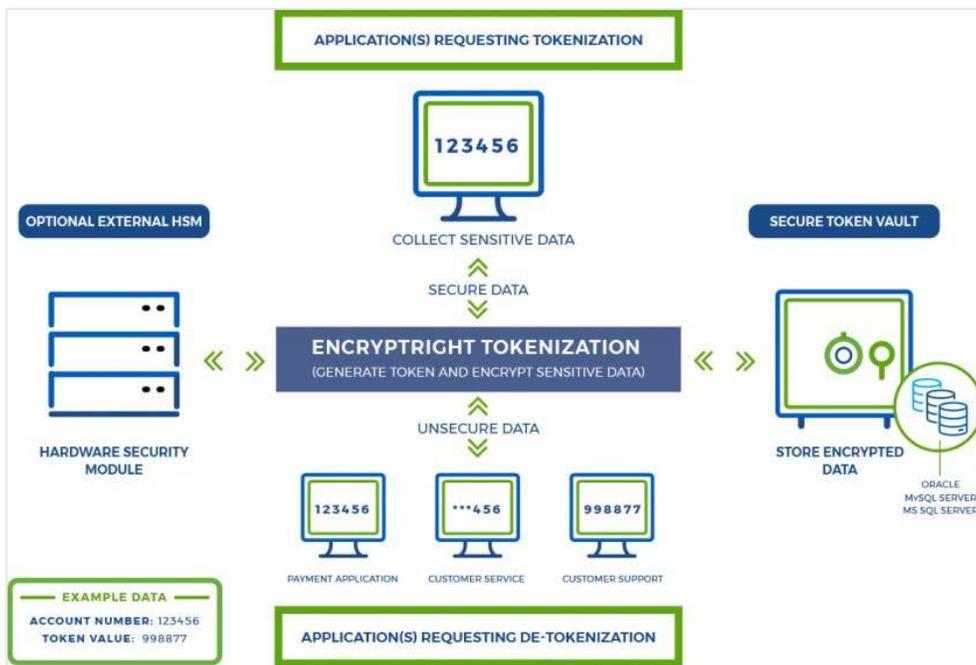


privacy needs. Tokenization solutions that can leverage role-based access controls, application-level data protection, and dynamic data masking to deliver broad data protection while automating data privacy management are poised to add the most value to enterprises concerned with meeting industry standards and data protection regulations.

Implementation Architectures

Tokenization architecture is typically approached from one of two general postures, loosely styled as “vaulted” and “vaultless.” Vaulted tokenization leverages a secure token database or “vault” to store the original data (which should be encrypted before being stored in the token vault) along with the corresponding tokens that represent the data, such that each token can be cross-referenced or mapped to its originating sensitive data. Vaultless tokenization derives a token from a mathematical algorithm in a manner similar to the way encryption works. When the original data is required, the tokenizing algorithm can be used to restore the original data from the token (or “detokenize”), without the use of the mapping inherent to a token vault – as such a token vault is not required. In a typical deployment, the algorithms and cryptographic keys used in vaultless tokenization are shared with the various systems leveraging vaultless tokens, so that these systems can detokenize data without needing to reference a centralized token vault.

The use of a token vault tends to offer the most flexibility in terms of token types and generation techniques, such as the use of random number or quantum *random number generators* (RNG) to



create tokens or leveraging encryption techniques similar to those used in vaultless tokenization. When algorithms are not used to generate tokens, as is the case in RNG tokens, there is no risk of cryptographic keys being cracked or algorithms being used to independently detokenize sensitive information. Because all detokenization required in vaulted tokenization is done

within the secure token vault, there is never a need to distribute tokenizing algorithms or cryptographic keys across the enterprise or cloud for detokenization. This is why the traditional method of leveraging a token vault is considered to be the most secure tokenization method.

The typical argument for vaultless tokenization is superior performance derived from decentralizing tokenization functionality, placing it closer to the applications leveraging tokenization without needing to return to a centralized token vault to access look-up tables (which can grow very large with enterprise deployments). However, vaulted tokenization benefits from the experience of the last decades and contemporary iterations have been able to demonstrate high-performance, large-scale, sub-second response, real-time transaction processing systems. Nevertheless, customers must often balance speed with flexibility and security when choosing a tokenization solution that makes the most sense for their business needs.

Vaulted Tokenization Deployment Architectures

Enterprises seeking the advantages of vaulted tokenization to reduce their PCI DSS audit scope/expense or for other data protection needs can choose between at least a couple of deployment architectures typical of contemporary technology: inside the network perimeter or outsourced to the cloud. Tokenization may be deployed only on in-house servers, supported by staff or well-vetted contractors, offering complete oversight of all the functions, equipment, logs, and procedures. Alternatively, enterprises might outsource the material elements of the architecture to cloud providers, whose economies of scale provide a lower cost basis (fixed costs are allocated across a number of customers).

Decisions in this area balance the cost to the enterprise and the confidence in the cloud provider, both in terms of operational excellence and integrity of their personnel. Outsourcing this type of mission-critical function must be reviewed in context of a cloud provider's proven performance — even a commitment of 99.9% uptime allows for outages over 8 hours in a year, and an outage of even a fraction of that time during critical periods could materially impact financial performance. Perhaps more important are risks associated with storing sensitive regulated data in an environment with an unknown number of “faceless” system operators, database administrators, and site integrity engineers who access sensitive data and may use it fraudulently⁹. With data protection regulations continuing to be put in place world-wide, business leaders face a growing challenge concerning liabilities associated with who owns the data and where the data takes residence, more commonly known as “Data Sovereignty.” While storage, management and protection of data can be outsourced to a third-party vendor or cloud provider, liability associated with a data breach cannot.

Range of System Support

Initial entrants in the market invariably reflected the competencies of the engineers designing and developing them, with examples oriented toward the open systems server platforms and Windows, while others prefer IBM mainframe z/OS-centric in their architecture. While one might excel in providing the needed tokenization capabilities for their native platform, many enterprises soon discovered that the workflows processing the sensitive data they sought to protect spanned mixed platforms. For example, a bank could adopt an open-systems server tokenization solution based on its close alignment with the incumbent consumer demand deposit banking package, only to find that the Social Security and account numbers being tokenized are shared with a mainframe mortgage loan processing system. In that scenario, having to replace the protective surrogate value with the original sensitive clear text value when crossing platform boundaries weakens the tokenization data protection model and, if a

second tokenization solution is adopted for the alternate operating system, complexity and associated costs grow.

The market demands that tokenization solutions support all the major enterprise computing platforms. Once a token is generated, it must be usable across all operating systems, transparently translating between the ASCII and EBCDIC encoding schemes, if needed, and supporting all the data types offered by every platform. The best tokenization implementations natively implement the needed capabilities on each platform, for best performance and maintainability, while remaining consistently interoperable.

The same principal applies to the range of relational database systems supported¹⁰. Early tokenization examples might only provide their own proprietary database system or, alternatively, align with one of the major general-purpose commercial database platforms (most frequently, Oracle or Microsoft SQL Server). The former offers advantages for reducing the time to implement, while introducing the burden of administration and security of another database system, while the second leverages existing administration and database protection procedures but introduce additional software license expense.

Current needs are met only when the tokenization implementation embraces both of the back-end database options above and expands beyond them. Less expensive relational database options, such as MySQL, have matured to the point of being viable for mission-critical production operations. The tokenization solution the market requires must have the flexibility to support all of these options.

Range of Token Generation Options

The first applications and tools for tokenization offered limited choices for creating the tokens, typically either random generation of a substitute value or encryption of the clear text input. Moreover, when encryption was the focus, the cryptographic source would be either software or hardware. Each addressed a particular need.

As noted above, randomly selected tokens are preferred by some Qualified Security Assessors (QSAs) evaluating payment card processing, as they more closely align with the standards published by the Payment Card Industry regulators. These tokens are generated using advance randomization techniques and each is assigned as the surrogate for the corresponding sensitive value, without any other implicit mathematical or algorithmic relationship. It is, essentially, an industrialized implementation of the concepts of a 'one-time pad' – a method of associating one surrogate value with another, using a random source, and ensuring that the surrogate value is never used again. When implemented correctly, such systems are essentially uncrackable by analysis or brute force, as there are no mathematical relationships to analyze.

In contrast, some systems elected to derive surrogate values using strong encryption algorithms. The values produced could not be easily cracked or reverse-engineered, so long as the cryptographic keys were protected from compromise. This gives the advantage of always being reversible. The issue of key protection further distinguishes categories of early implementations, in that two approaches were seen. One approach implemented encryption using only software, which offered greater flexibility and scalability with less durable key protection. The other approach implemented encryption using

hardware security modules (HSMs) which required channeling through those host-based appliances, offering the best cryptographic key protection available, though limiting scalability.

Finally, the original default of tokenization systems was to generate a new token for each incoming piece of sensitive data received and depend on the indexing of the tokenization tables to resolve them. This proves clumsy when the same sensitive value recurs frequently, such as the payment card account number a customer uses repeatedly at a grocery store. In these cases, tokenization system performance improves and ease of use goes up when “reusable” tokens are assigned, such that the same token is used as the surrogate for the unique instance of sensitive data, such as the payment card number mentioned above, or the SSN used to identify the treatment records of a patient at clinic.

Experience shows, however, that any single approach limits the situations in which tokenization can be applied. Enterprises need the flexibility of selecting the approach that is best for a given application or workflow, without the complexity of managing multiple tokenization solutions from multiple vendors. The best tokenization implementations support the full range of random token generation and encrypted token generation, using either software or hardware including support for a range of HSMs from a variety of vendors.

Range of Token Formats

Early implementations of tokenization focused on the primary function of creating, managing, and supporting surrogate values for corresponding sensitive data inputs. However, simply providing a surrogate value without regard to data type and character set severely limited the usefulness of the capabilities. Randomly generated or encrypted tokens with lengths or character sets inconsistent with those of the original sensitive data may fail edits when substituted for the original data.

Consider the example of the banking system in the Range of Use Cases section below. Each of the diverse programs that make up the system are constrained to expect a nine-digit numeric value, the format of an SSN, as the primary ID. If the random tokens used to replace them are 12 positions long, there is a risk of overwriting data in the following three positions of records and files. If an encrypted token is used, the encryption algorithm could produce an alphanumeric surrogate, and data integrity edits in the systems would reject them.

Practical application of early tokenization systems underscored the need to ensure that tokens could be specified that mimicked the numeric or alpha-numeric formats of the original inputs, replacing original characters with random values from the same character set with any special characters or spacing being preserved. This is known as “Format Preserving Tokenization”. Format Preserving Tokenization is a practical way of replacing sensitive data with surrogate values in existing systems and applications without needing to modify existing system edits and integrity checks, since tokens embody the same format as existing data.

As important as length and composition, controls must also allow for ensuring integrity checks implicit to the input values, such as modulus 10 checksums¹¹ required for electronic payment card numbers, are supported. Additionally, since these replacement tokens would so closely resemble the sensitive data inputs, there needs to be quick, automated, easy means to distinguish that a token is not an instance

of the type of data it is meant to replace (i.e., a token is a surrogate and not a valid payment card number, even though it is 16 digits long and passes the required modulus 10 checksum validation).

Some vendors offer the ability to specify token length or character types to replace a given piece of data, regardless of the size, character type, or even uniformity of the data-source. This is referred to as “Format Targeting Tokenization.” For example, you might tokenize a 9-digit SSN field using the format `***-**-****`, with a 12-character value using only alphabetical characters, or perhaps you have a set of account numbers that vary in size from 5 digits to 16 digits, but you would like every token to be a 16-character hexadecimal token.

Range of Use Cases

Many of the systems that require the additional data protection afforded by tokenization were not originally built with the thought that such requirements would be imposed. Some systems in the electronic payments and other industries have a foundation that is decades old and reflect design decisions that predate the pervasive information technology connectedness that now puts sensitive data at increased risk. The sensitive data may be used in discrete transactions, relational database updates, batch processes that add or modify records on the mainframe, user-driven ad hoc processes requesting mainframe control system responses, or open systems processes that generate output files. Initial tokenization systems might address one or even a couple of these typical use cases.

For practical, cost-effective implementation of tokenization, however, enterprises quickly identified that the capabilities must be equally available across all the enterprise systems (as noted above) and all of the workflows where sensitive data was in use. Additionally, integration of the capabilities must support the programming languages appropriate and familiar to each platform, natively implemented and with simple syntax to reduce the development and testing required to add the needed capabilities. The application programming interface (API) must support each of the different types of workflow use cases noted above, consistently implemented and fully supported.

For example, in the original design of a consumer banking system done late in the last century, before data breach and identity theft became so common, the designer might have elected to use SSNs as a primary key for mapping transactions, records, and accounts to a single customer. SSNs, which are sensitive personally identifying information, would then likely appear in all transaction postings to the demand deposit accounts¹² on the mainframe core banking package, in the unprinted control records of statement production files generated by a Windows Server-based print program, and included in the inputs loaded to an IBM i¹³ fraud risk analysis routine. However, now that sensitive data protection is governed by global, federal and state regulations, including the GDPR, the GLBA, and the CCPA, among many others, banks using the system must reduce the number of times clear text instances of the SSNs are used in processing. Designing a system and recoding all the programs on all the varied platforms would be an expensive and risky project, whereas integrating tokens in place of the SSNs and integrating API calls into the programs when the original value is needed in place of the surrogate token is far less intrusive and far less expensive. The cost is further reduced when the API offered consistently implements the capabilities in a manner well suited to each platform’s familiar programming languages, such as COBOL for IBM environments or C# and Java for Windows.

Range of Input Sizes

The need for support of all data types for all the major enterprise platforms is already discussed above, as are the range of data use cases that benefit from tokenization (fields in transactions or records, database columns, even entire files in some edge cases). While one might think these would cover all the expected inputs, initial iterations of tokenization quickly underscored the need for additional granularity in the definition of fields that need to be tokenized. When the only consideration is a 16-digit payment card number, supporting fixed length fields at fixed locations in transactions or records may be sufficient. However, practical experience showed that many workflows rely on variable-length fields of sensitive data that occur in variable locations within a transaction or record. Imposing constraints that previously variable length fields must maintain an arbitrary fixed length solely to comply with a tokenization strategy only introduced unneeded complexity and associated cost to enterprises.

For example, a payment card processing system originally designed before the recognition of the risk that cyber criminals represent uses a single workflow for processing transactions for Mastercard/Visa/JCB/Discover (all 16-digit card numbers) as well as American Express (15 digits). The account numbers for all the incoming transactions would benefit from tokenization but forking the inbound streams to sort by account number length would introduce unwarranted recoding and retesting to an existing, efficient process. Equally, attempting to work with padded fields so that every incoming transaction record is 'padded' with zeros to arbitrarily fit the maximum length introduces other complications. The better approach takes the variability of the field into account, supplying the token needed based on the length of the input value.

Relevant examples of tokenization encompass this flexibility, determining the appropriate token to associate in the incoming account number regardless of its length. As important, it extends the flexibility to multiple variable-length fields in a variable-length transaction or record, with the same agility.

Range of Roles

The arc of technology development is consistent, regardless of the discipline. First efforts focus on proving that the innovation actually works and provides the intended benefits, followed quickly by keen focus on improving operational efficiency, in terms of manufacture, installation, and operation. As noted in the beginning of this discussion, only after these first two goals are addressed does attention turn to issues of safety and security. The market has arrived at this point in its demands of tokenization, requiring that the role of the security administrator, setting policy for protection of sensitive data by means of tokenization, be segregated from the roles of developers integrating the needed capabilities into application workflows.

This requirement satisfies customer best practice to ensure that no one person both establishes controls for protecting data and applies the controls to the sensitive data that needs to be protected – a situation that offers many opportunities for abuse. It also reduces the cost of integration by reducing the familiarity integration programmers must have with the complex fields of data security, key management, and tokenization. The tokenization administrator can focus and specialize in that discipline, without having to learn the subtleties of coding in C, Java or COBOL, while developers can specialize in integration for a given environment.

Fitting Tokens to the Need

The several discussions and examples in this paper outline the capabilities that enterprises must expect from tokenization offerings sufficient to meet current and anticipated needs:

- ◆ Aligns exactly with the constraints imposed by relevant standards and regulations (e.g., PCI DSS, GDPR, GLBA, CCPA, HIPAA).
- ◆ Deploys in-house, for maximum control of system availability, or in the cloud, as needed.
- ◆ Implements capabilities consistently across all the major enterprise computing platforms and specifically accommodates all file and data types, regardless of platform.
- ◆ Includes quick-to-market integration, implementing APIs suitable for each platform.
- ◆ Supports all the major enterprise relational database systems.
- ◆ Offers a range of token generation capabilities, including random generation & assignment plus key generation and management from either software or hardware cryptographic sources.
- ◆ Treats one or many variable length fields of sensitive data in variable length records.
- ◆ Anticipates the need for reusable tokens for frequently received sensitive data values.
- ◆ Imposes a separation of duties between the security professional defining data protection policies and the professional developer integrating tokenization capabilities into existing or new applications.

With these capabilities in place, an enterprise's risk of data breach, cost of regulatory audit and compliance, and ongoing cost of data protection is lower than when using less flexible alternatives.

For an expanded discussion of the capabilities included in leading tokenization technology, please contact Prime Factors.

ENDNOTES

¹ Source: 2018 Thales Data Threat Report, Thales and 451 Research

² Source: Gartner: Security of Applications and Data Primer for 2019

³ <https://eugdpr.org/>

⁴ <http://en.wikipedia.org/wiki/GLBA>

⁵ https://en.wikipedia.org/wiki/California_Consumer_Privacy_Act

⁶ <https://www.insideprivacy.com/international/brazils-new-general-data-privacy-law-follows-gdpr-provisions/>

⁷ https://en.wikipedia.org/wiki/Health_Insurance_Portability_and_Accountability_Act

⁸ https://en.wikipedia.org/wiki/Health_Information_Technology_for_Economic_and_Clinical_Health_Act

⁹ <https://www.cnn.com/2019/07/29/business/capital-one-data-breach/index.html>

¹⁰ Almost without exception, contemporary vaulted tokenization systems rely on relational database systems for managing the cross-reference tables mapping original sensitive values to surrogate tokens.

¹¹ Also known as mod 10, or a Luhn check: <http://en.wikipedia.org/wiki/Luhn>

¹² i.e., checking and savings accounts

¹³ Also known as AS/400, IBM iSeries, etc.