Choosing Tokenization or Encryption

By Jeff Stapleton – ISSA member, St. Louis Chapter

This article discusses the similarities and differences between two popular cryptographic techniques: tokenization and encryption. When making the decision between protection methods, there are several things to consider, including how the data is used and the key management life cycle.

A common question is how to choose between encryption and tokenization. Unfortunately such a simple question is rather difficult to answer. An information security professional might respond with “it depends,” which is actually quite reasonable but sadly unhelpful. Both techniques are within the cryptography domain and share similarities and have differences. This article compares both techniques in an effort to help answer the conundrum: when to use encryption and when to use tokenization. Both protect data via confidentiality, but neither provides integrity or authenticity [10].

Encryption

First things first, to discuss between tokenization versus encryption, definitions for each need to be established. Everyone know what encryption is, or at least most have a concept of the technology, but ironically there are various similar but slightly different definitions.

1. Encryption1 is the process of encoding a message or information in such a way that only authorized parties can access it and those who are not authorized cannot.

2. Encipherment: rendering of text unintelligible by means of an encoding mechanism [1].

3. Encryption is the (reversible) transformation of data by a cryptographic algorithm to produce ciphertext (i.e., to hide the information content of the data) [5].

4. Cipher: Series of transformations that converts plaintext to ciphertext using the cipher key. [4]

Regardless of which encryption definition seems more familiar or makes the reader the most comfortable, a basic description of encryption is the use of cryptographic keys to transform data ( cleartext) from a readable to an unreadable (ciphertext) form. However, this is only half an explanation. Decryption is the inverse of encryption, the use of cryptographic keys to transform ciphertext back to its original cleartext. Figure 1 provides a graphical view of symmetric encryption and figure 3 shows a similar depiction of asymmetric encryption. Symmetric encryption uses the same (secret) key for encrypt and decrypt functions, whereas asymmetric encryption uses two different keys, the public key for encrypt and the private key for decrypt.

Figure 1 shows Alice using the symmetric encryption function with two inputs, the cleartext and the secret key, and one
output, the ciphertext. Similarly, Bob uses the symmetric decrypt function with two inputs, the ciphertext and the secret key, and one output, the cleartext. When it is just Alice and Bob, they need to establish a shared key; but when multiple parties are involved, there are key management considerations.

<table>
<thead>
<tr>
<th>COMMON SHARED KEY</th>
<th>UNIQUE KEY PER PAIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Key B</td>
</tr>
<tr>
<td>Chuck</td>
<td>Key B</td>
</tr>
<tr>
<td>Dave</td>
<td>Key B</td>
</tr>
<tr>
<td>Alice</td>
<td>Key A</td>
</tr>
<tr>
<td>Chuck</td>
<td>Key C</td>
</tr>
<tr>
<td>Bob</td>
<td>Key D</td>
</tr>
</tbody>
</table>

Figure 2 – Symmetric Key Management

Figure 2 shows symmetric key management when multiple parties are involved. The left side shows when Bob shares a common “Key B” with Alice, Chuck, and Dave. However, cleartext encrypted by Alice might be decrypted by Chuck or Dave, so there is a distinct lack of confidentiality or privacy. The right side shows when Bob establishes a unique key per pair. Alice and Bob use “Key A,” Chuck and Bob use “Key C,” and Dave and Bob use “Key D.” Consequently, cleartext encrypted by Alice, Chuck, or Dave can be decrypted by Bob but not by the other parties. Nevertheless, Bob must manage a symmetric key for each party, which is not particularly scalable. Asymmetric cryptography can reduce the number of keys that need to be managed.

Figure 3 illustrates Alice using the asymmetric encryption function with two inputs, the cleartext and Bob’s public key, and one output, the ciphertext. Conversely, Bob uses the asymmetric decrypt functions with two inputs, the ciphertext and his associated private key, and one output, the cleartext. The public key is mathematically derived from the private key, but the private key cannot be derived from the public key, so Alice cannot access Bob’s private key. Public keys might be used as either a data encryption key (DEK) for data encryption or a key encryption key (KEK) for key management. Alice needs to validate that the public key be-

---

2 This article does not address quantum computer risks (QCR) or post-quantum cryptography (PQC).

3 Note that only reversible asymmetric algorithms (e.g., RSA) can be used for encryption.
longs to Bob. Public keys are typically managed using digital certificates that are signed and issued from a certification authority operating within a public key infrastructure (PKI), but such a discussion [11] is beyond the scope of this article.

Figure 4 shows asymmetric key management options. As discussed for asymmetric encryption, Bob’s public key might be used to encrypt the data. An alternative might be to use Bob’s public key⁴ to encrypt a random key (RK) and then use the random key to encrypt the data. A random key would be unique per message (not just per pair). Other asymmetric key management schemes (e.g., key agreement) are beyond the scope of this article.

### Tokenization

Likewise, everyone seems to know what tokenization is, or at least has an opinion. Further, the term *token* means different things to different people. Some interpret a token to be a physical device used for "something you have" authentication. Others use *token* to mean a cryptographic value (e.g., digital signature) used within a security protocol (e.g., SAML message). However, for this article, a token is a substitute data element used instead of the actual data element. Depending on the application domain, tokenization can actually have slightly different meanings. Similar to encryption, tokenization has several definitions.

1. **Tokenization**⁵ when applied to data security, is the process of substituting a sensitive data element with a non-sensitive equivalent, referred to as a token that has no extrinsic or exploitable meaning or value.

2. **Tokenization or Tokenize**: The process of mapping a plaintext value (i.e., the USV⁶) to an existing or newly generated surrogate value (i.e., a token) [2].

3. **Tokenization** is a process by which the primary account number (PAN) is replaced with a surrogate value called a token. Note that this definition is specific to the Payment Card Industry (PCI) tokenization standards [7][8][9].

4. **Tokenization**: A process by which the primary account number (PAN) is replaced with a surrogate value called a payment token. Tokenization may be undertaken to enhance transaction efficiency, improve transaction security, increase service transparency, or to provide a method for third-party enablement. Note that this definition is specific to the EMV payment tokenization standard [3].

Regardless of which tokenization definition seems more familiar or makes the reader the most comfortable, a basic description of tokenization is the use of a cryptographic method to substitute readable cleartext with other readable cleartext. This is analogous to old-fashioned codes when word substitution was used. For example, a spy might report three ducks and two frogs in the pond, but the covert message would be three destroyers and two submarines in the harbor. The purpose of tokenization is to avoid disclosure of sensitive data by using tokens. Figure 5 provides a graphical representation of tokenization, token vaults, and detokenization processes.

1. The tokenization function has two inputs, the cleartext and the token methodology (Random, Table, Encrypt and MAC,⁷), and one output, the token. Each of the methods is described with more detail in the X9.119-2 standard [2].

   a. **Random method**: uses a random number generator (RNG) to generate a random value for the token.

   b. **Table method**: uses an RNG and pseudo-random number generator (PRNG) to generate a static table used to generate tokens.

   c. **Encrypt method**: uses a symmetric key to encrypt the cleartext to generate a token.

   d. **MAC method**: uses a symmetric key with a message authentication code (MAC) algorithm and the cleartext to generate a token.

---

³ Note that irreversible asymmetric cryptography (e.g., Diffie-Hellman) can be used for key agreement.


⁵ Underlying Sensitive Value (USV).

⁶ Message Authentication Message (MAC) or keyed-Hash Message Authentication Code (HMAC).
2. The token vault can be used with any token methodology to match or verify cleartext and tokens, as the vault is a database of related pairs.

3. The detokenization function has two inputs, the token and token methodology (inverse Table, Decrypt), and one output, the cleartext.

   a. **Random method**: the cleartext cannot be recovered from the token as the two have no determinable correlation.

   b. **Table method**: the inverse function (Table') is used to recover cleartext from the token.

   c. **Decrypt method**: uses a symmetric key to decrypt the token to recover the cleartext.

   d. **MAC method**: the cleartext cannot be recovered from the token as it does not provide sufficient data to derive the cleartext.

Note that the Random and MAC methodology cannot be used for detokenization. Further, some tokenization solutions use more than one method. For example, a hybrid tokenization might employ the Table with Encrypt methods for tokenization and Decrypt with inverse Table methods for detokenization.

Hence, encryption is a recognized tokenization method, which adds to the confusion. Further, some tokenization solutions incorporate format preserving encryption (FPE) [15]. FPE is not a cryptographic algorithm per se, rather it is a mode of operation that uses an existing encryption algorithm (e.g., AES) with cleartext and symmetric key as inputs, but its ciphertext maintains the length and character set as the cleartext. For example, if a six-digit number is encrypted using FPE, the output is another six-digit number. The FPE modes are reversible such that the ciphertext can be decrypted. Figure 6 provides several tokenization examples for purposes of discussion.

Suppose some online service provides four applications (Movie, Coupons, Zodiac, and Game) that incorporates birth dates. But the provider has determined that birth dates are sensitive information that need to be tokenized. The first and second columns in the table show birth dates for Alice, Bob, Chuck, and Dave. The third column shows the tokenized birth dates stored in a tokenization vault.

- **Movie**: this application needs to determine the user’s exact age by checking the birth date against the current date. The movie application might submit the token (e.g., 2023) for detokenization so it can access the complete birth date. Alternatively, the movie application might employ its own token (e.g., 2441338) shown in the fourth column that might be detokenized or processed in lieu of an actual

<table>
<thead>
<tr>
<th>Name</th>
<th>Birth Date</th>
<th>Token</th>
<th>Movie</th>
<th>Coupons</th>
<th>Zodiac</th>
<th>Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1/22/1972</td>
<td>2023</td>
<td>2441338</td>
<td>72201</td>
<td>9022</td>
<td>2122</td>
</tr>
<tr>
<td>Bob</td>
<td>2/13/1975</td>
<td>2025</td>
<td>2442821</td>
<td>71302</td>
<td>9045</td>
<td>2013</td>
</tr>
<tr>
<td>Chuck</td>
<td>5/8/1964</td>
<td>2013</td>
<td>2438523</td>
<td>70805</td>
<td>9128</td>
<td>1908</td>
</tr>
<tr>
<td>Dave</td>
<td>11/6/2001</td>
<td>3017</td>
<td>2452229</td>
<td>70611</td>
<td>9310</td>
<td>1906</td>
</tr>
</tbody>
</table>

**Figure 6 – Tokenization examples**


Choosing Tokenization or Encryption | Jeff Stapleton

- **Coupon**: this application needs the user’s approximate age for issuing discounts. The coupon application might submit the token (e.g., 2023) for detokenization so it can access the whole birth date, but alternatively, the movie application might employ its own token (e.g., 72201) shown in the fifth column. The coupon token might be partially detokenized to get just the year, or the token might be processed in lieu of an actual birth year. This might isolate the coupon tokenization process from other applications.

- **Zodiac**: this application needs the user’s month and day for astrological information. The zodiac application might submit the token (e.g., 2023) for detokenization so it can access the whole birth date. But, another approach might be a separate token (e.g., 9022) for the zodiac application, shown in the sixth column. The zodiac token might be partially detokenized to get just the month or date, or the token might be processed directly. This might isolate the zodiac tokenization process from other applications.

- **Game**: this application employs the user’s day of the month to start the game. The game application might submit the token (e.g., 2023) for detokenization so it can access the whole birth date, but a separate token (e.g., 2122) might be used to isolate the game application from the other games and tokens.

As discussed, the applications might share a common tokenized birth date that needs to be detokenized, but this puts the whole online service environment at risk. Each time a token is detokenized it increases the likelihood of abuse or compromise of the tokenization system. Employing application-unique tokens helps isolates the applications and protects the tokenization system from misuse.

**Comparison**

Encryption is intended to protect data in storage or during transmission. For storage, ciphertext is stored on disk, decrypted, and the cleartext is processed by an application. For transmission (see figures 1 and 3) ciphertext is transmitted, decrypted, and the cleartext is processed by an application. The encryption commonality is that cleartext is processed by an application.

Tokenization is intended to protect data in storage or when processed. For storage, tokens are stored on disk, read, and processed by an application. For transmission, tokens are transmitted, received, and processed by an application. The tokenization commonality is that tokens are processed by an application, but for applications that cannot process tokens, the tokens are detokenized.

As Luther Martin observed in his Crypto Corner: “Encryption vs. Tokenization” [6]:

*There now seems to be a battle between security vendors over the relative benefits of encryption and tokenization that seems to be very similar to the sectarian division between the Big-Endians and the Little-Endians that Jonathan Swift described in Gulliver’s Travels. While Lilliputian scriptures clearly said that an egg should be broken at the convenient end, this did not stop partisan factions from resorting to violence to enforce their will on those with the other point of view. Encryption and tokenization are more similar than they are different. And just like it made sense for Lilliputians to open the convenient end of an egg first, enterprise security departments should use whichever of...*
Choosing Tokenization or Encryption | Jeff Stapleton

the two technologies solve their particular problems best. Despite what some vendors may tell you, tokenization is actually equivalent to a form of encryption. Once you start looking at the way that either tokenization or encryption might be used in an enterprise environment, a reasonable rule of thumb seems to emerge: if you want to protect data that will rarely get unprotected, tokenization may be the best approach, but if you want to protect data but may have to unprotected it to enable further processing of it at some point, encryption may be the best approach. There are cases where each technology is superior to the other so that there is no single best technology for all applications. Anyone who tells you otherwise is selling you something.

Data might be encrypted or tokenized to protect it during storage. However, unless the data is randomized in some fashion, the same data will result in the same ciphertext or token. For instance, in the tokenization examples, the same birth date (e.g., 2/13/1975) for different names will result in the same token (e.g., 2025). Thus, an attacker whose birth date is included in an application has an attack vector (cleartext, ciphertext) pair, and the attacker knows the birth date for any matching ciphertext.

Data size can be an issue for encryption or tokenization. Very small data cannot always be properly protected. Consider a single character field with only two possible values. Regardless of the encryption algorithm or key length, two cleartexts can only result in two ciphertexts unless the cleartext or algorithm is altered per encryption. Here are a few possible alternatives:

- **Concatenate multiple cleartext fields for encryption.** For example, instead of separately encrypting the five small data fields AA, BB, CCCC, DDDD, and EEEE, encrypt the concatenated field AABBCCCCDDDDEEE.

- **Randomly pad cleartext fields for encryption.** For example, instead of encrypting the small data field AA, encrypt the padded field AAZXYKMLWRGHUOSX.

- **Use random encryption algorithm parameters.** For example encrypting AA with the same cryptographic key but with a random initialization vector or random tweak results in a unique ciphertext for each encryption.

- **Use random encryption keys.** For example encrypting AA with a random cryptographic key results in a unique ciphertext for each encryption.

However each of these alternatives has operational issues such as affecting the decryption process, increasing the work factor, storing parameters, or managing multiple keys. Tokenization has a similar “small data” problem: only two possible cleartext can only result in two tokens, except for the random method where the same cleartext tokenized ten different times generates ten different tokens.

Conversely, very large data might exceed an encryption algorithm’s recommended maximum cleartext or the tokenization’s maximum cleartext capacity.

Encryption methods tend to use the same algorithms for reliability and simplifying implementations, but different cryptographic keys strengthen the overall security. However, applications relying on tokenization need uniformity, so the algorithms (and keys) need to be consistent. Tokenization methods using the same algorithms and the same keys (or keyless) weaken the overall security.

Both techniques have design and deployment decisions that can be documented within a cryptographic architecture [12][14]. Similar to network diagrams or application data flows, a cryptographic architecture illustrates where, how, and when cryptographic keys are deployed and employed. It can identify where data is encrypted or tokenized, how the keys are stored, and when ciphertext is decrypted or tokens are detokenized. Keys used for encryption or tokenization within a cryptographic module might be handled in software or protected by a hardware security module [13].

**Conclusion**

Thus, the question when to use encryption or tokenization really has an “it depends” answer. Things to consider include whether the data is being protected from unauthorized access when stored, when being transmitted, or when being processed. Luther also points out that the frequency of being unprotected (decrypted or detokenized) is another consideration. Another aspect is whether the protected data needs to be unique per instance or can the same ciphertext or token reflect the same cleartext. Further, does the length or format of the protected data need preserving, and if so, then the protection solution might need to employ format preserving encryption (FPE). And finally the data life cycle versus the key life cycle needs to be understood. Changing cryptographic keys for encryption necessitates translating ciphertext from a previous key to a newer key, but changing keys or tables when using tokenization also requires replacing older tokens with newer tokens. The key management life cycle might have enormous impact on the applications relying on the data protection methods.

**References**


A software crisis followed as the demand of software projects outweighed the supply of programmers. In 1968, NATO met to solve the problem of the shortage of programmers. The most significant outcome of the meeting was changing the term “programming” to “software engineering” to remove the stereotype of the type of worker and the pay associated with it. This changed the professional status, hiring practices, and certification programs through the ’60s and into the ’70s.

With new intrigue from the renamed title of software engineer, the addition of educational prerequisites contributed to the masculinity of the profession. It became harder for women who might have been self-taught programmers or stay-at-home moms to get employment. In other words, what began as “women’s work” without educational prerequisites was now shifting to men’s work. While women had on-the-job software engineering experience and skills, they began to interpret this shift as exclusion. Coupled with the more important factors of wage discrimination and lack of child care and mentoring support, women began to avoid the industry altogether [2].

By 1970, the ARPANET packet-switching network linked computers together, leading to the Internet of today. It’s design and use was primarily male dominant because of the military funding and work by mathematicians, computer scientists, and engineers at universities who were fixing bugs and writing protocols for transferring data; however, early hypertext was designed by women. In the early years of the Internet, personal computers attracted women back to the field and a new generation of female workers was born. In fact, the ARPANET directory and resource handbook for shared users were written by a female team, headed up by a woman named Jake.

Fast forward to today. There are ample cybersecurity and technology jobs, and ways to fill the pipeline are on everyone’s mind. Getting there takes work and awareness to move the needle, but many are seeing a slow but sure shift.

Conclusion

The wartime years reflected a computer age dominated by women. Women embraced technology and shaped the foundation for innovation. Software and hardware developments caused the workforce to shift toward male dominance. While, today’s environment is much different, we are still overcoming generations past. Given time, history will turn another 180.

References


About the Author

Dr. Curtis C. Campbell is VP of Atlantic Capital Bank in Atlanta, GA, and chapter president of ISSA Chattanooga. She is a cybersecurity author with 25 years experience in information security, compliance, procurement, and third-party risk in the enterprise. She serves on the advisory board of University of TN-Chattanooga, a national Center for Academic Excellence for Cyber Defense (CAE-CD) studies. Connect with Curtis via curtis@mprotechnologies.com.