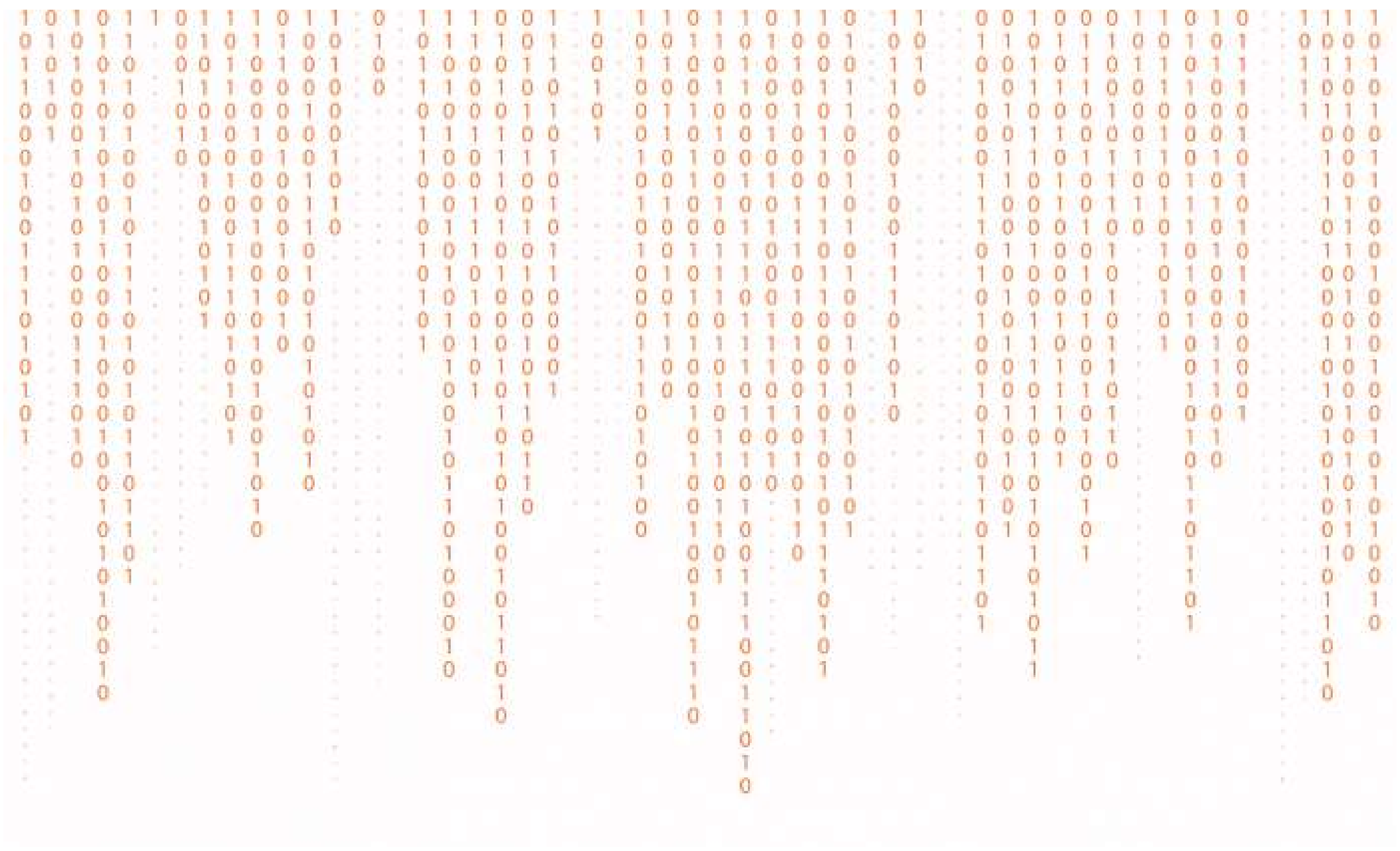


The difference between Encryption, Hashing and Salting

(4 votes, average: 4.00 out of 5)

FACEBOOK TWITTER LINKEDIN MAIL

December 19, 2018

47

The difference between Encryption, Hashing and Salting

Encryption and Hashing both serve different functions despite their similarities

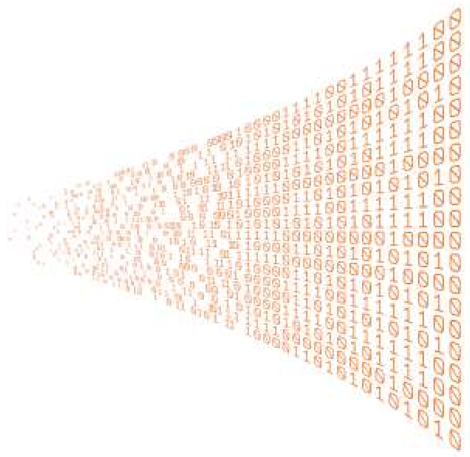
Quick, do you know the difference between encryption and hashing? Do you know what salting is? Do you think salting your hash is just part of an Irish breakfast?

All kidding aside, if you pay any attention to the world of cybersecurity you're likely to hear these terms bandied about. Oftentimes without any explanation.

So, today let's talk about the difference between encryption and hashing – and answer any questions you may have been too afraid to ask. Along the way we'll also cover salting, since it's in the news almost every single time a password database gets compromised.

Let's hash it out.

What is Encryption?



Encryption is the practice of scrambling information in a way that only someone with a corresponding key can unscramble and read it. Encryption is a two-way function. When you encrypt something, you're doing so with the intention of decrypting it later.

This is a key distinction between encryption and hashing (forgive me the pun).

To encrypt data you use something called a cipher, which is an algorithm – a series of well-defined steps that can be followed procedurally – to encrypt and decrypt information. The algorithm can also be called the encryption key. I realize the word algorithm has kind of a daunting connotation because of the scars we all still bear from high school and college calculus. But as you'll see, an algorithm is really nothing more than a set of rules–and they can actually be pretty simple.

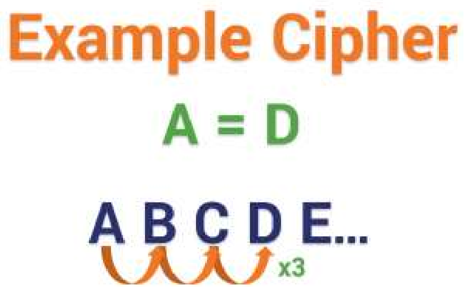
Encryption has a long, storied history. It dates back to at least 1900 BC after the discovery of a tomb wall with non-standard hieroglyphs chiseled into it. Since then there have been countless historical examples.

The ancient Egyptians used a simple form of encryption. As did Caesar, whose cipher stands as one of the most important examples of encryption in history. Caesar used a primitive shift cipher that changed letters around by counting forward a set number of places in the alphabet. It was extraordinarily useful though, making any information intercepted by Caesar's opponents practically useless.

A couple thousand years later, a nomenclator cipher – a type of substitution cipher that swaps symbols for common words in an attempt to avoid a decryption technique called frequency analysis – got Mary Queen of Scots beheaded and a bunch of conspirators killed when their messages were intercepted and deciphered by a literal man (men) in the middle.

How does Encryption Work?

Let's take a look at encryption using a simple cipher. We'll take a page out of Caesar's playbook and go with a shift cipher. I'm going to encrypt a sentence using a mono-alphabetic shift cipher that simply replaces each letter with one that is sequentially three places ahead of it.



Obviously we've come a long way since the early days, but just focus on the concepts. The sentence in its raw form is in plaintext – unformatted, unencoded – the state it will return to once it has been decrypted.

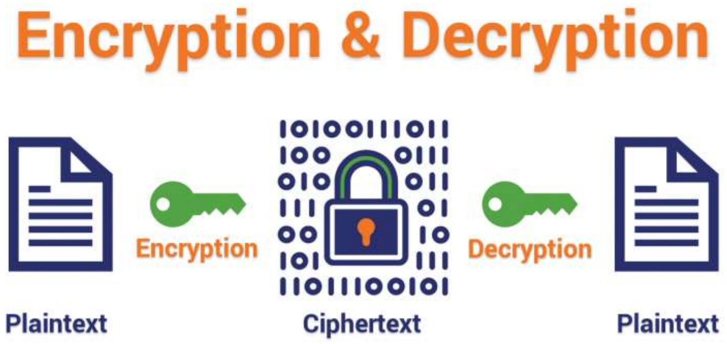
Let's go with the sentence, "don't be a jerk," which, incidentally, is also the only rule for our comments section. I'm going to apply the encryption algorithm/key, and turn it into ciphertext.

Plaintext: Don't be a jerk

Becomes:

Ciphertext: Grqwehdmhun

I've omitted the punctuation for the sake of simplicity but ciphertext is usually transmitted without spacing or punctuation anyway to avoid errors and hide word boundaries. So, provided I've shared it with them, some could now use the corresponding key – in this case the inverse of the algorithm that was used for encryption – to decrypt this message and read it. Obviously, the ciphers we use in digital encryption are much more complex, but you get the general concept behind it.

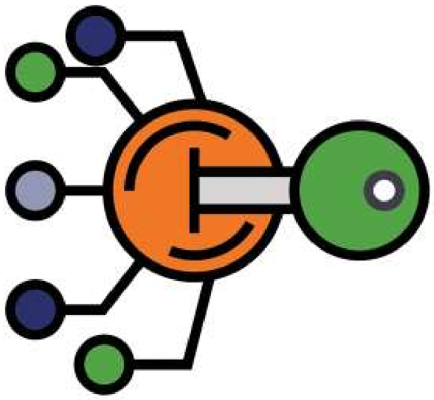


Historical encryption algorithms

Let's start by going over some different types of ciphers, then we'll get into the modern algorithms that are used in today's encryption.

Shift Ciphers – Like the example we discussed above, two parties determine a number between 1-25, and shift the letters that number of spaces in the alphabet. The shift number serves as the key.

Substitution Ciphers – These ciphers replace plaintext with cipher text using an algorithm that is a fixed system. The key is the document that shows the fixed system, which can be used to reverse engineer the encryption.



Transposition Ciphers – This algorithm uses a set of rules, which serve as the key, to change the order of the text into a different permutations that can then be encrypted. Common examples are Rail Fence and Route ciphers.

Polyalphabetic Ciphers – These are a type of substitution cipher that use multiple alphabets to further complicate unauthorized decryption of the ciphertext.

Nomenclator ciphers – A type of substitution cipher that replaces common plaintext words with symbols to try and throw off a specific form of cryptanalysis.

Cryptanalysis is the study of cryptosystems with intention of finding weaknesses in them. One of the most common forms of cryptanalysis, that dates back to an Arab mathematician named Al-Kindi who lived around 800 AD, is called frequency analysis. It examines the ciphertext for repetitive symbols or strings of characters and cross references them with words that would appear with a high frequency in the message that's being decrypted.

So, for instance, if you're writing a message to Napoleon, it's only logical that you would use his name a few times. By matching the ciphertext version to his name, it helps you to start mapping the key and decrypting the message.

Polyalphabetic ciphers and nomenclator ciphers were better suited to withstand frequency analysis than their classical counterparts. Polyalphabetic ciphers continued to be used until World War II when the Enigma machine was cracked.

Modern Encryption

Before we can talk about modern encryption ciphers, we need to talk a little bit about public and private keys and how the digital revolution has changed encryption. All of the examples that we just went over are what we call Private Key Cryptography. Encryption was entirely contingent upon a private key, which had to be physically exchanged in order for decryption to take place. If you know anything about Private Keys, it's the fact that they are sacrosanct. Having your Private Key compromised can be disastrous. So having to physically carry and pass it along only makes it that much more of a risk. People have actually died over private key compromises throughout history.

Today, thanks to computer technology and the internet we can now practice public key cryptography. With public key cryptography, one public key is used to encrypt and the other private key is used to decrypt. You see this during the SSL handshake, where they've solved the historically risky issues with physical key exchange by using the publicly available key to encrypt a symmetric session key and send it back to the server for decryption by its private key. Well, you don't actually see it, but you catch my drift.

Today, the most common forms of encryption are:

- **Asymmetric Encryption** – This is the Public Key example we just gave. One key encrypts, the other key decrypts. The encryption only goes one way. This is the concept that forms the foundation for PKI (public key infrastructure), which is the trust model that undergirds SSL/TLS.
- **Symmetric Encryption** – This is closer to a form of private key encryption. Each party has its own key that can both encrypt and decrypt. As we discussed in the example above, after the asymmetric encryption that occurs in the SSL handshake, the browser and server communicate using the symmetric session key that is passed along.

Between the two, asymmetric encryption tends to be stronger owing to its one-way nature.

When you're shopping for an SSL certificate and see “2048-bit” tossed around, that's in reference to private key length, specifically an RSA private key. When you see “256-bit” mentioned, that's typically referring to the size of the symmetric session keys that are used during the actual communication. That doesn't mean symmetric encryption is less safe. It would still take a supercomputer thousands of years to decrypt 256-bit encryption.

The reason symmetric 256-bit encryption is used to communicate is that it's faster, which means better performance and less overhead for servers.

Taking a Closer Look at the SSL/TLS Handshake

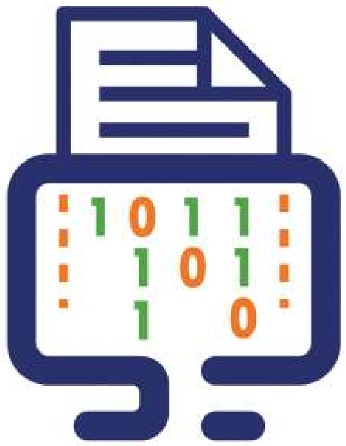
In Everything Encryption • By Patrick Nohe

There's a lot going on underneath the hood when you connect to a website via HTTPS. First and foremost, everyone needs to… shake hands?!

Read more

Modern Encryption Algorithms

Now that's we've discussed symmetric and asymmetric encryption, we can get into some modern encryption algorithms.



AES – AES stands for Advanced Encryption Standard, originally called Rijndael, it's the specification for encryption published by the National Institute for Standards and Technology (NIST) back in 2001. It puts plaintext through a number of "transformation rounds" determined by key size, each round consists of several processing steps. Let's not stray too far into the weeds on this one. AES is a common algorithm with SSL/TLS. It replaced the Data Encryption Standard (DES) that was created in 1977.

RSA – RSA stands for Rivest-Shamir-Adlemen, after its creators, it is a public key encryption algorithm (asymmetric) that has been around since 1978 and is still widely used today. It uses the factorization of prime numbers to encipher plaintext.

[Fun Fact: The unfortunately named Clifford Cocks, a mathematician employed by the GCHQ, a British intelligence agency, invented an equivalent system five years earlier, in 1973, but it wasn't declassified until 1997.]

ECC – ECC stands for Elliptic Curve Cryptography, which relies on the algebraic structure of elliptical curves over finite fields. Although ECC has been around since 1985, it's only been in use since about 2004. ECC has distinct advantages over RSA and is likely going to play a more prominent role in the future of SSL/TLS.

PGP – PGP stands for Pretty Good Privacy, it was created in 1991 by Phil Zimmerman. It's really more of a collection of algorithms than a single one, all for hashing, data compression and both public and private key cryptography. Each step uses a different algorithm. PGP has been criticized for poor usability, a lack of ubiquity and for the length of its keys.

When should encryption be used?

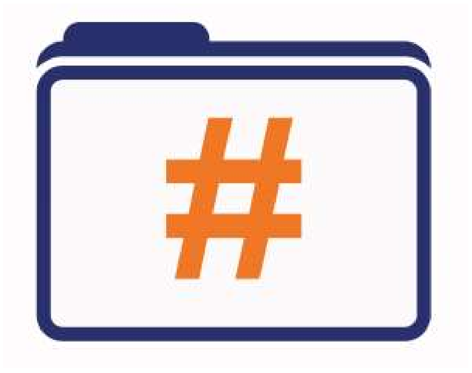
As we discussed earlier, encryption is a two-way function. You encrypt information with the intention of decrypting it later. So, correspondence with someone online, protecting your cloud data or transmitting financial data are all examples of times when encryption is appropriate.

The key is that encryption is reversible. Hashing is not.

What is Hashing?

Hashing is the practice of using an algorithm to map data of any size to a fixed length. This is called a hash value (or sometimes hash code or hash sums or even a hash digest if you're feeling fancy). Whereas encryption is a two-way function, hashing is a one-way function. While it's technically possible to reverse-hash something, the computing power required makes it unfeasible. Hashing is one-way.

Now, whereas encryption is meant to protect data in transit, hashing is meant to verify that a file or piece of data hasn't been altered—that it is authentic. In other words, it serves as a check-sum.



Here's how it works, each hashing algorithm outputs at a fixed length. So for instance, you may hear about SHA-256, that means that the algorithm is going to output a hash value that is 256 bits, usually represented by a 64 character hexadecimal string (h/t Matthew Haslett).

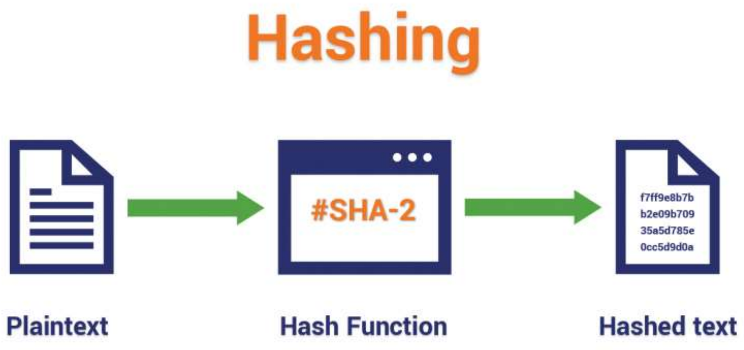
Every hash value is unique. If two different files produce the same unique hash value this is called a collision and it makes the algorithm essentially useless. Last year, Google created a collision with the SHA-1 hashing algorithm to demonstrate that it's vulnerable. SHA-1 was officially phased out in favor of SHA-2 in early 2016. But Google had a point to make so it devoted two years' worth of funds, man hours and talent in a partnership with a lab in Amsterdam to make something that was to that point more of an abstraction into a reality. That's a long way to go to prove a point. But Google went there.

Anyway, here's an example of hashing, let's say you want to digitally sign a piece of software and make it available for download on your website. To do this, you're going to create a hash of the script or executable you're signing, then after adding your digital signature you'll hash that, too. Following this, the whole thing is encrypted so it can be downloaded.

When a customer downloads the software, their browser is going to decrypt the file, then inspect the two unique hash values. The browser will then run the same hash function, using the same algorithm, and hash both the file and the signature again. If the browser produces the same hash value then it knows that both the signature and the file are authentic—they have not been altered.

If it's not, the browser issues a warning.

That's actually how code signing works. Just remember, no two files can create the same hash value, so any alteration – even the tiniest tweak – will produce a different value.



Common Hashing Algorithms

Just like we did with encryption, let's take a look at some of the most common hashing algorithms in use today.

MD4 – MD4 is a self-loathing hash algorithm, created in 1990, even its creator, Ronald Rivest, admits it has security problems. The 128-bit hashing algorithm made an impact though, it's influence can be felt in more recent algorithms like WMD5, WRIPEMD and the WHSA family.

MD5 – MD5 is another hashing algorithm made by Ray Rivest that is known to suffer vulnerabilities. It was created in 1992 as the successor to MD4. Currently MD6 is in the works, but as of 2009 Rivest had removed it from NIST consideration for SHA-3.

SHA – SHA stands for Security Hashing Algorithm and it's probably best known as the hashing algorithm used in most SSL/TLS cipher suites. A cipher suite is a collection of ciphers and algorithms that are used for SSL/TLS connections. SHA handles the hashing aspects. SHA-1, as we mentioned earlier, is now deprecated. SHA-2 is now mandatory. SHA-2 is sometimes known has SHA-256, though variants with longer bit lengths are also available.

RIPEMD – A family of cryptographic hashing algorithms with a lengths of 128, 160, 256 and 320 bits. It was developed under the framework of the EU's Project Ripe by Hans Dobbertin and a group of academics in 1996. Its 256 and 320 bit variants don't actually add any additional security, they just diminish the potential for a collision. In 2004 a collision was reported for RIPEMD-128, meaning RIPEMD-160 is the only algorithm from this family worth its salt (this is going to be an amazing pun in about two paragraphs).

WHIRLPOOL – Designed by Victor Rijmen (the co-creator of the AES algorithm we discussed earlier) and Paulo Barreto in 2000. Since then it has undergone two revisions. It produces 512-bit hashes that are typically represented as 128-digit hexadecimal numbers.

TIGER – A fairly new algorithm that is beginning to gain some traction with file sharing networks and torrent sites. There are currently no known attacks that are effective against its full 24-round variant.

What is Salting?

Salting is a concept that typically pertains to password hashing. Essentially, it's a unique value that can be added to the end of the password to create a different hash value. This adds a layer of security to the hashing process, specifically against brute force attacks. A brute force attack is where a computer or botnet attempt every possible combination of letters and numbers until the password is found.

Anyway, when salting, the additional value is referred to as a "salt."

The idea is that by adding a salt to the end of a password and then hashing it, you've essentially complicated the password cracking process.

Let's look at a quick example.

Say the password I want to salt looks like this:

```
7X57CKG72JVNSSS9
```

Your salt is just the word SALT

Before hashing, you add SALT to the end of the data. So, it would look like this:

7X57CKG72JVNSSS9SALT

The hash value is different than it would be for just the plain unsalted password. Remember, even the slightest variation to the data being hashed will result in a different unique hash value. By salting your password you're essentially hiding its real hash value by adding an additional bit of data and altering it.

Now, if a brute force attacker knows your salt, it's essentially worthless. They can just add it to the end of every password variation they're attempting and eventually find it. That why the salt for each password should be different – to protect against rainbow table attacks [h/t Jon's suggestion in the comments].

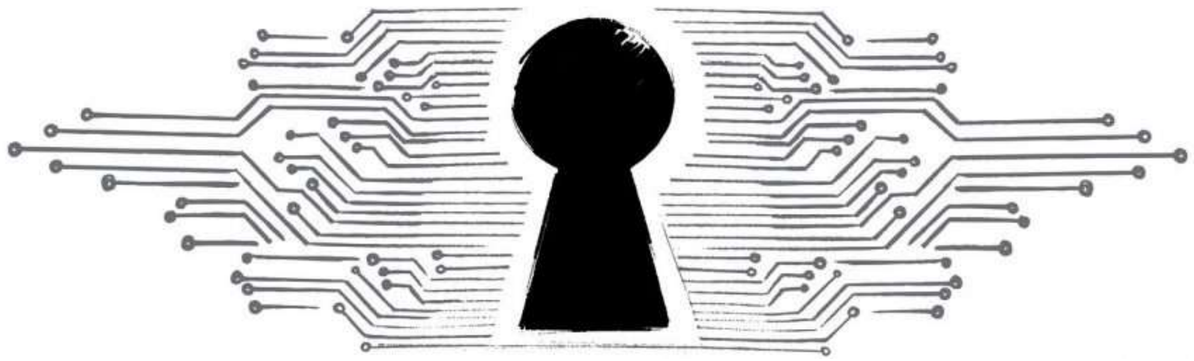
We could write an entire article on password security (which we now have) and whether it's still even a useful safeguard, but for now that should be a passable definition of salting.

And a quick aside, if you hadn't put two and two together by now, our name, Hashed Out, is a play on the popular idiom for discussing something and the hashing process involved in SSL encryption. *Hey Patrick, that's really clever*. Thank you for noticing.

What we Hashed Out (for Skimmers)

Here's what we covered in today's discussion:

- Encryption is a two-way function where information is scrambled in such a way that it can be unscrambled later.
- Hashing is a one-way function where data is mapped to a fixed-length value. Hashing is primarily used for authentication.
- Salting is an additional step during hashing, typically seen in association to hashed passwords, that adds an additional value to the end of the password that changes the hash value produced.



#ENCRYPTION #HASHING #SALTING

Author

Patrick Nohe

Patrick started his career as a beat reporter and columnist for the Miami Herald before moving into the cybersecurity industry a few years ago. Patrick covers encryption, hashing, browser UI/UX and general cyber security in a way that's relatable for everyone.

Search

Search

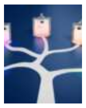
Recent Posts

How to Sign a Word Document Using a Digital Signature Certificate
February 23, 2024

5 Ways to Avoid Your Company Falling for Deepfake Scams
February 7, 2024

How Do I Make My Website Secure? The Essential Guide
February 2, 2024

The Ultimate Guide to 13 U.S. Data Privacy Laws (And What They Mean to Your Business)
January 29, 2024



Demystifying SAML Authentication: A Look at the Role of SAML in SSO-Based Access Management
January 19, 2024

Follow Us

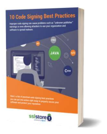
Free Ebooks

Email Spoofing Defense 101 (A 5-Step Guide)

Download Now

Certificate Lifecycle Management Best Practices Guide

Download Now



10 Code Signing Best Practices

Download Now

Your email address

Subscribe to Hashed Out

Buyer Zone

Extended Validation Cert
Domain Vetted Cert
Organization Certificates
Server SSL Certificates
Email & Documents Signing
Free Tools
Compare SSL Certificates

Partner With Us

Partner Program Overview
Reseller Program
Affiliate Program
API & Integrations
WHMCS Module
AutoInstall SSL
Strategic Partnerships

About Us

About Us
Blog
SSL Clients
Case Studies
Why Choose Us

24/7 Help Zone

SSL Support
Manage Your Account
FAQ
Help with EV
Site Map
Contact Us

★★★★★ The SSL Store 8,967 Ratings & Reviews SHOPPER APPROVED

Request for Quotation Custom Integration Resources SSL Installation Service

The SSL Store™ | 146 2nd Street North #201 St. Petersburg, FL 33701 US | 727.388.1333
© 2024 The SSL Store™. A Subsidiary of DigiCert, Inc. All Rights Reserved.

Privacy Policy Disclaimer Refund Policy