SAS3034-2019

How SAS® Viya® Ensures that Your Data Is Safeguarded by Default with Secure Encryption Techniques

Alec Fernandez, Destiny Harrell, SAS Institute Inc., Cary, NC

ABSTRACT

What is "data in motion"? What techniques does SAS® Viya® use to protect your data in motion from unwanted inspection or tampering? Why should you care? This paper introduces the concepts behind securely moving data across networks without fear of a data breach or data falsification. First, there's an overview of Transport Layer Security (TLS), x.509 certificates, cipher algorithms, and SAS® Secret Manager. I explain how SAS® Viya uses TLS to establish secure communications between SAS servers and clients. I describe how SAS generates, distributes, and renews certificates, and how SAS delivers and manages the list of trusted Certificate Authorities. I also explain how you can replace certificates provided by SAS Viya with your own certificates for any servers that users access directly. From a security perspective, I show how to disable and enable network security. In summary, I describe how SAS deploys software onto multiple machines while ensuring an end-to-end secure environment by default. Come learn how your data in motion is kept safe and secure.

PREFACE

The study of information security makes you skeptical. At first, your skepticism is limited to knowledge obtain through hearsay. For example, "Bob sent me a message across the internet telling me that Alice has a bridge for sale." You instinctively doubt this information. This level of skepticism is common. Most adults have learned to be skeptical of hearsay. You might reasonably ask, "What if this is wrong? What if Bob simply misheard Alice?"

But after further study, your skepticism increases. You begin to wonder, "What if Bob is lying?", "What if this message didn't actually come from Bob? What if the message that Bob got from Alice about the bridge wasn't really from Alice? What if it actually came from that malcontent Malory? What if Mallory was impersonating Alice and it was she who made up the whole thing about the bridge?" This could mean that Bob was tricked into sending you the message.

Besides trust, there are other issues. Knowing what you can believe is only part of the problem. For example, what if Alice and Bob don't want you (or Eve) to know about the bridge? How can they keep that information private? What if Alice sells you the bridge and then denies having received your payment. How can you prove you paid her?

These are all very real scenarios that happen regularly.

The more you study, the more you reach the unsettling conclusion that when it comes to information you receive over a computer network, you must safeguard and confirm information very carefully.

In this paper we will introduce terms to better understand the scenario described above and how to ensure that information that you send and receive over computer networks is private and reliable.

INTRODUCTION

All information sent through the internet is sent much like postcards through the postal service. It is visible to anyone who is looking whether or not they want to see it.

Consider Wireshark. It is a very simple program. It displays whatever network traffic reaches your computer. Anyone can download and install it and use it to observe network traffic. It's very useful and is frequently used to diagnose server connection problems.

I recall using Wireshark to debug a program I was working on that was supposed to accept a network connection. I was watching the packets of bytes that were reaching my computer. As expected, many of these packets that I was seeing were not addressed to my computer and I could see them being discarded by the network interface. Among these packets that were destined to be discarded, I could see my boss's user ID and password being transmitted as he was trying to log in to another machine on the network. These were displayed in clear text. I could not help seeing them. I was not explicitly snooping but you cannot avoid seeing what is placed on a screen in front of you. My boss was not explicitly aware either, he simply typed his password on the keyboard and pressed enter and did not stop to consider how it was being transmitted. Similarly, all manner of private information: bank website credentials, credit card numbers, love notes, anatomical photos are flying all over the internet every instant of every day. We need ways to keep this private information secured even when there are many people explicitly snooping.

You probably already have an intuitive sense of why privacy is important, but it is helpful to have some specific terms to describe the benefits we gain from secure communication:

Integrity: The message has not been altered. You want to pay $100, not $100,000.
Authentication: The recipient of the message can verify the identity of the author. You know the offer to sell you goods for a set price is genuine.
Confidentiality: The message can only be received by authorized persons, not the snoopers.
Non-repudiation: The person sending the message cannot repudiate (deny) having been the author. The sender cannot deny having agreed to buy goods by claiming forgery.

To better understand the concepts, it is helpful to understand the historical context in which the field emerged.

HISTORY OF CRYPTOGRAPHY

The use of encryption goes back to the earliest days of recorded history. As can be inferred from the word "crypt", the concept of encryption means to store information in a "location" where it is safe. Information can be kept safe in plain sight if it is stored in a format that renders it inaccessible unwanted intruders. Ciphers were used by craftsmen in 1500 BCE Egypt to conceal secret recipes for pottery glazes so that their competitors could not steal the recipes that allowed the inventors to charge higher prices for their work[1].

[1] (History of Cryptography, n.d.)

Ciphers are simply techniques of scrambling information in a way that makes it unintelligible. The cipher used by Julius Caesar to keep his correspondence (and likely the recipe for Caesar Salad Dressing) secret works as follows:

Take every letter in the plaintext message and replace it with the letter that occurs 4 letters earlier in the alphabet

This would render the text "the quick brown fox jumps over the lazy dog"
as "qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald"

This cipher requires both parties to know the alphabet and a secret. The secret would be the number of letters to shift, 4 in this case

To decipher the message, you simply reverse the cipher algorithm and you get the plain text back.

As with Caesar, the military has traditionally been a very active consumer of cryptography. Their need to securely transfer information across open networks affects the security of individuals and national governments. The same principles of security apply whether you are writing an encrypted message on parchment before handing it to a horseman for delivery through enemy territory or whether you are sending messages across the internet. Many of the problems associated with keeping secrets that were discovered early on in the realm of data security, still apply today.

- The message might be captured by the enemy. Can the enemy decipher it?
- The messenger might be compelled to reveal the cipher. Can the messenger decipher it?
- The messenger might be complicit with the enemy. Can the messenger alter the message in a way that cannot be detected by the recipient?

Clearly in the case of the Caesar cipher, if the adversary were to discover the cipher algorithm, he could ascertain the meaning of the message and all past messages.

The use of cryptography to protect valuable commercial property provided strong incentive to adversaries to defeat these protections and steal the valuable information that was being transmitted. This could be commercially valuable trade secrets or militarily valuable battle plans

The first documentary evidence of the use of crypto analysis to break ciphers appears around AD 800. This made evident several problems inherent in encryption.

- There is no way to send a message to a single individual that cannot be seen by any messenger that has the cipher algorithm and encrypted text
- Once you crack the cipher, you can see all messages encrypted with that cipher for all of history. If you anticipate that a cipher will be eventually disclosed or cracked or if you want to send private messages to different individuals you need many ciphers
- It is very hard to know if a cipher is secure. Each of us can devise an algorithm that is beyond our own individual ability to crack it. History has shown that very few of us are willing or able to devise algorithms that teams of other individuals cannot crack. Many algorithms are secure enough for individuals or companies because these are unwilling to hire a large staff of people willing to do tough math calculations for years to break a code. But it when governments marshal vast resources to crack a code, it can be done.

Once the science of crypto analysis came into existence, the use of clever devices to encrypt messages was rendered obsolete. Even early people concluded that simple approaches such as alphabet shift ciphers were no longer effective.
But it took several centuries of escalations to reach a conclusion. For centuries, very clever individuals devised new encryption techniques that were fiendishly difficult to crack. These would be used until very clever individuals eventually discovered even more sophisticated methods to crack them.
An important turning point came in 1883 when Auguste Kerkhoffs wrote two journal articles in La Cryptographie Militaire[2] in which he cited 6 design principles for military ciphers. One still stands: "It should not require secrecy, and it should not be a problem if it falls into enemy hands."

Put another way, the cipher itself should not be secret. Everything about the cipher should be public except for one piece of information, the encryption key. Without this secret, the message is secure.

Why is this important?

Suppose an individual were to invent an encryption machine/method/algorithm that could be proven to be secure. Cryptographers could be given unlimited quantities of both plaintext messages and their encrypted equivalents and still not be able to crack the encryption algorithm. This algorithm would certainly serve it's intended purpose. People would be able to communicate securely without fearing that eavesdroppers, no matter how sophisticated, motivated and well-funded, could crack the encryption technique.

But in practice there is a big problem. Anyone with access to the algorithm itself would be able to see all messages. And to sell the algorithm, the inventor must convince the buyer that it works. To do this, the inventor must divulge the approach to someone who understands and who is trusted by the buyer. As soon as this happens, at least two people now know the algorithm. At best, this means that at the secret is harder (impossible?) to keep. For the buyer to get this number back down to the number of informed individuals required for secret keeping (one), it would end up being a bad day for the inventor.

More generally, if the only thing an adversary must do to crack an encryption technique is to understand the algorithm, this has proven to happen with regularity in practice. Secrets are difficult to keep.

- Bribe or capture and coerce the inventor
- Steal the algorithm
- Reverse engineer the algorithm
- Wait until someone else re-invents the same algorithm
- Read WikiLeaks
- Listen to Edward Snowden

The fact that it is impossible to keep an algorithm secret is what led to the acceptance of Kerkhoffs's principle.

So, what does an encryption machine that conforms to Kerkhoffs's principle look like?

[2] (Kerckhoffs's Principle, n.d.)

Figure 1 Encryption Machine

Public Domain[3]

An encryption machine, or cipher, takes a message and a secret key as inputs. It encrypts the message and produces encrypted symbols as output. It is impossible to derive the original text from the encrypted output. When provided with the same secret key and encrypted symbols as input, the same machine can perform the inverse process and produce the original text as output.

[3] Image from https://commons.wikimedia.org/w/index.php?curid=51169837.

Now comes the hard part: You give this machine to adversarial governments, to friends, to academics, to the public. You allow them to take the machine apart and you openly describe how it works. In software systems, the machine is simply an algorithm. You make this algorithm public.

You give the public access to both the plaintext and the encrypted versions of many messages. You let them create their own messages and secret keys so that they can devise special messages and keys specifically intended to show weaknesses in the encryption techniques.

If anyone can crack the algorithm, you go back to the drawing board.

World War II provided a tremendous incentive to make improvements and many advances were made in the field of encryption and cryptanalysis. It's reasonable to assume that there might have been advances that were not made public during this time. But despite this gargantuan effort, insofar as we know, the problem of finding a way to conceal data without the risk of clever remained unsolved stood for a very long time after the end of the conflict.

THE FIRST PUBLIC ENCRYPTION STANDARD

The search for a secure, publicly available encryption standard continued until Horst Feistel showed the way forward. Horst was born of Jewish descent in Germany in 1915. He grew up with the backdrop of deteriorating social conditions for Jews in Germany. In 1934 he emigrated to the United States and obtained degrees from MIT and Harvard. Subsequent to this, his stay in the US was made less pleasant by being place under house arrest during World War II due to his being of German descent. Despite this obstacle, he obtained US Citizenship in 1944 and the very next day he was granted security clearance and was employed by the US Air Force where he worked on methods to distinguish from friends from foes in communication systems.

It's easy to see how this type of work might be helpful decades later in devising schemes for securing computer networks.

He invented the "Feistel Scheme" for using block ciphers (rather than stream ciphers) to encrypt data while working at IBM in the 1970s. A refined version of this scheme was submitted by his employer to the National Bureau of Standards (NBS). In 1976, the NBS collaborated with the National Security Agency (NSA) to alter the algorithm in ways that made it stronger against secret cryptoanalysis attacks that were known only to the NSA. But it was made weaker against brute-force attacks.

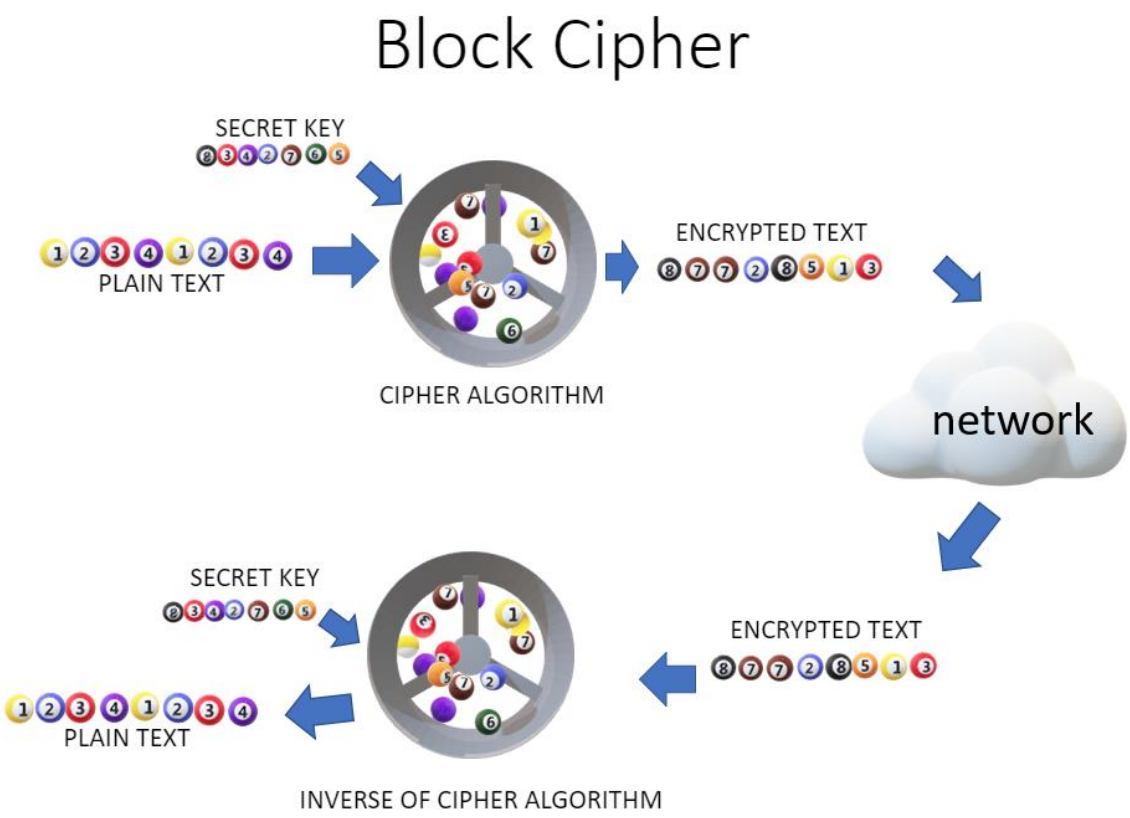


Figure 2 Feistel Scheme Block Cipher Technique

At the time of its release, DES received contemporary criticism from academics for having an excessively small (56 bit) key size. They explained that the math was not very daunting and that if computers continued to advance at the same rate as was occurring in the mid-1970s, it was possible that they would someday be fast enough to defeat the algorithm by simply trying all possible values of the key (a brute force attack). Despite this criticism, it was adopted by the NBS and was dubbed "THE" Data Encryption Standard or DES. It was published as an official Federal Information Processing Standard (FIPS) Pub 46 in 1976 and gained widespread use.

The DES algorithm proved that the combination of a public cipher and a randomly selected private key could be used to openly transmit messages securely and that these messages would be private to holders of the secret key.

Much has been reported about the motives behind the NSA's selection of the key size about their desire to balance the mission of providing security to the United States and their obligation to respect privacy laws. It took 22 years (an instant in the field of mathematics), for the academics to be proven correct by publicly breaking DES in 1999. The calculations required to reverse engineer the key by trying all possible combinations took a mere 22 hours and 15 minutes.

This serves to underscore the fact that the competition between cryptography and cryptanalysis is alive and well.

But the Feistel scheme is made of durable stuff. It wasn't quite defeated. You can increase the effective key size with this simple approach:

1. Pass the plain text input through the DES algorithm using a 56-bit key
2. Use the encrypted output from step 1 as input to the DES algorithm but supply a different key (Key-2).
3. Use the "double encrypted" output from step 2 as input for a third pass through the DES algorithm but supply yet another different key (Key-3)

This "triple encrypted" output from step 3 is Triple-DES or 3DES and can only be decrypted if you have all three keys[4].

As of 2019, Triple DES is still widely used today in the financial industry. It remains reasonably secure in practice despite the existence of theoretical attack vectors. However, it should be abandoned as quickly as possible because the margin for error is very thin and other vulnerabilities can be combined to exploit 3DES in ways that defeat.

The combination of a public encryption algorithm (DES) and a strong commercial incentive to secure internet traffic (e-commerce) drove rapid advances in the fields of cryptography and cryptanalysis. The fact is we have known almost since the outset that DES would someday fall, and much work has been done in anticipation of this event. In 1997, the NIST began a 5-year long, open contest to select a better encryption algorithm to replace DES.

Two Belgian cryptographers, Vincent Rijmen and Joan Daemen, submitted a block cipher algorithm built on the techniques pioneered by Feistel. In 2001, their algorithm was selected as the most suitable proposal. This algorithm is the Advanced Encryption Standard (AES). It is included in the ISO/IEC 180333-3 standard. It is approved for use by the NSA and the meets the latest FIPS-140 specifications[5].

AES encryption with, a suitably long random key, gives you the ability to share information in a completely secure manner. All that is required is that all parties to the conversation have the same key.

ASYMMETRIC ENCRYPTION

As you now know, Symmetric Key Encryption requires all parties to the conversation use the same secret key.

This leaves one big limitation and a few problems. Suppose you want to have a secure conversation with an individual or a bank website that you have never previously met. This entity cannot possibly know your secret key. Without a shared key, you cannot communicate. How do you provide them with your secret key? You cannot send it over an unsecured, public medium like the internet because then everybody would know the key and you have lost security. You need a secured channel through which you can transfer the secret key. How do you pass a secret to an intended communication partner while everyone on the internet is carefully listening as it happens? Any bit of information you send to your partner of your impending secret communication will be seen by everyone.

DIFFIE-HELMAN (AND MERKLE)

[4] (Triple DES, n.d.)

[5] (Advanced Encryption Standard, n.d.)

As with ciphers in general, the rather vexing problem of bootstrapping a private conversation in a public setting stood unsolved in the public domain for millennia. You simply had to communicate shared secrets as best you could. You communicated them via verbal whispers, you wrote them on scraps of paper destined to become tasty morsels. You recorded them on cassette tapes destined to self-destruct in 5 seconds. None of these methods are secure.

Enter Bailey Whitfield Diffie (MIT) and Martin Hellman (University of California at Berkeley). Their 1976 paper *New Directions in Cryptography* set forth a new method of distributing secret keys that stands as the basis of asymmetric, public key, cryptography. Armed with the Diffie-Helman key exchange approach[6], you now have a way to securely share a random secret key with a person you have never met. Once you have shared this key, you can use AES to securely share messages with this person.

RSA

The Diffie-Helman solution required the use of a one-way function, a hash, that had not yet been invented. This problem was being studied the very next year at MIT by Ron Rivest, Adi Shamir, and Leonard Adleman. Rivest and Shamir were cryptographers proposing secure algorithms, Adleman was a mathematician who was proving that they were not, in fact, secure. This continued until Rivest had made a breakthrough that stumped Adleman. Indeed, the RSA problem continues to stump mathematicians to this day. Their research led to the publication of *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* in 1978 and to the Rivest-Shamir-Adleman (RSA) cryptosystem[7].

This cryptosystem uses two different but mathematically related keys that work as a perfectly matched set. One of the keys is used for encryption and the matching partner key is used for decryption. The encryption key is shared with the public, but the matching decryption key is kept private. A message encrypted with the public key cannot be decrypted unless you possess the corresponding decryption key and it is impossible to ascertain or derive the matching private key from any known information.

As set forth by Kerkhoffs, the algorithm behind RSA is public and easy to find. It is based on this formula:

$$Z = Y^x * Mod(N)$$

To get a sense of the power behind the math, consider the following:

- If X, Y and N are each 2,048 bits long Y^x consists of around 2^{2059} bits. That is a lot of bits. The Universe, by comparison, is thought to contain around 2^{250} particles. Hence, storing the result of this calculation is not feasible.
- You can avoid computing Y^x by applying the modulus function after each multiplication. This would require performing 2^{1900} 64-bit multiplications. On the fastest computers, this would take a significant timespan. A timespan many times the expected lifespan of our sun.

It is always good to have physics on your side.

This following scenario illustrates how it works in practice. Bob wants to send a message to Alice via RSA.

[6] (Diffie Hellman key exchange, n.d.)

[7] (RSA (cryptosystem), n.d.)

1. Alice generates two random keys. She keeps one of them private.
2. Bob obtains Alice's public key via a reliable, but not necessarily secret, route. For now, we let Alice send the public key across the internet. Everybody on the internet, including Bob, receives it.
3. Bob feeds Alice's public key and his message into the RSA algorithm, which generates encrypted text.
4. Bob sends this encrypted text across the internet. Everybody, including Alice, receives it.
5. Alice uses her private key and the RSA algorithm to decrypt the encrypted text and can see Bob's message. Everybody else is kept completely in the dark.

Obviously, Bob can do the same as Alice, send her his public key and they can communicate in both directions. This is the basis for every encryption algorithm used by SAS and for secured internet traffic.

You now have everything you need to carry out secure message exchanges in a public forum.

TLS

One drawback to the RSA algorithm is that it is computationally intensive. The Diffie-Hellman algorithm, because it uses a symmetric key (the same key is used to encrypt and decrypt) requires less computational effort and is faster. To optimize the speed of exchange information, RSA is used to establish a secure communication channel and this channel is used to agree upon a shared key. This key is then used to exchange information using a symmetric key algorithm.

Because all this negotiation of algorithms and exchanging of keys (and key lengths and ciphers, etc) happens under the prying eyes of the internet, it is imperative that it be done completely securely. Because of the commercial incentives provided by emergent internet commerce sites in 1995, it was imperative that it be done quickly. Briefly, a commercial company named Netscape released Secure Socket Level (SSL) protocol in 1994. Another commercial company, Microsoft, released a competing protocol. Within a year, these protocols were discovered to have security issues.

After many iterations of renaming, hardening and standardization, you now have TLS v1.2 that is widely believed to be secure.

X.509 CERTIFICATES

You can extend the principles of RSA to create a digital signature. A digital signature is a mathematical scheme for verifying the authenticity of a document. Digital signatures allow us to have very strong reason to believe that a document was created by a known sender.

The premise is that if you know the following, the message must have been encrypted (sent) by Alice[8].

- A message can be decrypted with Alice's public key
- Alice's one and only matching private key is the only thing that could have encrypted the message.
- Only Alice knows her private key

[8] (Digital signature, n.d.)

Now we have 4 of the major tenets of security covered:

- Confidentiality and integrity through unbreakable encryption algorithms and ciphers
- Authentication and non-repudiation: through digital signatures

We can standardize and extend these concepts to a digital certificate, identity certificate, or more specifically an x.509 certificate. X.509 was initially issued in 1988 to solve the problem of digitally validating identities[9].
The concept of an identity rests on trust. If an unknown person claiming to be Alice approaches you with an offer to sell a bridge, should you trust her? Anyone can claim to be Alice.

Now suppose Alice produces a document, a US passport. This document ties together two pieces of information (CN stands for Common Name)

- `Subject: "CN = Alice"`
- `Issuer: "CN =The US Department of State"`

This establishes a chain of trust. You do not inherently trust Alice, but you know that the entity that issued the passport, the Department of State, trusted Alice enough to have issued her the document. If you can trust this issuer, then you can also trust that the person named in the document: Alice. You will trust other information present on that same passport document: Issuance and expiration dates, other personal information such as the address and whether the person is a diplomat, the passport number, nationality, etc.

How do you know that this passport is not a forgery? In the physical world, there are various techniques, involving the holograms, u-v ink, RFID chips, etc. If you look very carefully at the document, you should be able to tell if the document is fake or if the original information was altered in some way. In the digital world, detection of forgeries is easier, the issuer digitally signs the document, so forgeries and alternations are impossible.

Suppose you have never heard of the issuer, the US Department of State. How can you know if the issuer is trustworthy? US Department of State could establish its trustworthiness by presenting a certificate of trust as follows:

- `Subject: "CN = The US Department of State"`
- `Issuer: "CN = The United States of America"`

Again, if you can trust the issuer, then you trust the subject. In this case if you trust the US government, then you can trust the US Department of State

Suppose that again, you ask to see the certificate of this issuer. That certificate would have the following:

- `Subject: "CN = The United States of America"`
- `Issuer: "CN = The United States of America"`

In this certificate, the subject is citing itself as the only source of authority attesting to the identity of the subject. There is no higher level of authority to attest to the validity of

- this certificate
- certificates signed by this certificate
- descendent certificates of certificates issued by this certificate

This certificate is the root of trust, the topmost level of authenticity.

[9] (X.509, n.d.)

This certificate is a trust anchor. In general parlance, it is also referred to as a Certificate Authority, a CA certificate, a root CA, or any similar, loose amalgamation of these terms. Since the issuer is the same as the subject, this certificate is also said to be a self-signed certificate. That is, it is signed with its own keys.

The certificate we described above, the one held by Alice herself, is referred to with terms such as leaf certificate, identity certificate, and end-entity certificate.

The list of certificates (there can be many) that exist between the trust anchor and the leaf certificates are said to form the certificate chain, certificate path, or chain of trust

There is no shortage of terms used to describe these certificates, but they are generally referred with terms such as intermediate certificate.

CERTIFICATE AUTHORITIES

Suppose you decide not to purchase this bridge that Alice portends to sell after all. You've heard amazon.com sells them cheaper.

You still need to know that you can trust the identity of the merchant. Certificate Authorities help with the process of establishing trust on the internet and help prevent man-in-the-middle problems. A man-in-the-middle attack works as follows:

If you want to establish a secure connection with amazon.com, you send a request across the internet that you hope reaches amazon.com. But some imposter intercepts this request. The imposter does 2 things. First, he responds to you with his identity certificate which your browser accepts and uses to establish a secure connection with the imposter. Soon thereafter, you expect to see your browser render the amazon.com home page. The second thing the imposter does is establish his own connection with amazon.com. Amazon.com is a public site and happily accepts this connection from the imposter which to amazon is simply one more happy customer. The connection between amazon.com and the imposter is secured exactly as you would expect. The imposter now has two connections, one with you and one with amazon.com. The imposter gets the amazon.com home page from amazon and sends you a copy. You get the amazon.com home page from the imposter thinking that it is amazon. You ask for the login page. The imposter receives your request and sends a copy of it to amazon.com. Amazon.com sends the imposter the login page he just requested and he forwards you a copy. You get the login page you were expecting. You are still assuming that you are communicating securely with amazon.com, so you interact with the login page. You have every reason to believe that you are interacting with amazon.com. Everything looks exactly the same because it's a copy. The imposter gets your responses (including your username and password) and forwards them to amazon, when the imposter gets the response from amazon, he sends you a copy. You choose an item and send back your payment information. The imposter records these and then forwards them to amazon. In the end, everybody gets what they want. You get your goods delivered from amazon and the imposter gets your login ID and payment information. Unfortunately, you got a lot more than what you bargained for.

In order to prevent these man-in-the-middle attacks, you need a way to verify that you are connecting to the merchant and not to a clever imposter. You need a way to securely verify the merchant's identity.

In the case of amazon.com, the merchant is a machine. More specifically, it is a web application server process running on a machine. The certificate presented to you by this

web application server is often called a "server cert" because it is used to identify a web server. The identity information in this certificate has to do with the name of the machine and the machine's aliases and perhaps IP addresses.
It will appear like this:

- `Issuer: "CN = DigiCert Global CA G2 O = DigiCert Inc C = US"`
- `Subject: "CN =` www.amazon.com `O = Amazon.com, Inc. L = Seattle S = Washington C = US"`
- `Subject Alternative Name: "DNS Name=buckeye-retail-website.amazon.com DNS Name=huddles.amazon.com DNS Name=p-nt.www-amazon-com.kalias.amazon.com`

You probably noticed that the Issuer of this amazon.com certificate is DigiCert Inc. DigiCert is a Certificate Authority.

A certificate authority (CA) is an entity that issues digital certificates. A digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others to rely upon signatures or on assertions made about the private key that corresponds to the certified public key. A CA acts as a trusted third party—trusted both by the subject of the certificate and by the party relying upon the certificate[10].

Certificate authorities serve you by only issuing (digitally signing) certificates for entities that can prove they possess the identity named in the certificate. If you are presented with a certificate with the common name (CN) amazon.com and the issuer is a trusted CA (DigiCert), then you can trust that the owner of the cert really is amazon.com.

It is natural for you to wonder why you should trust DigiCert as an authority worthy of attesting to the identity of amazon.com.

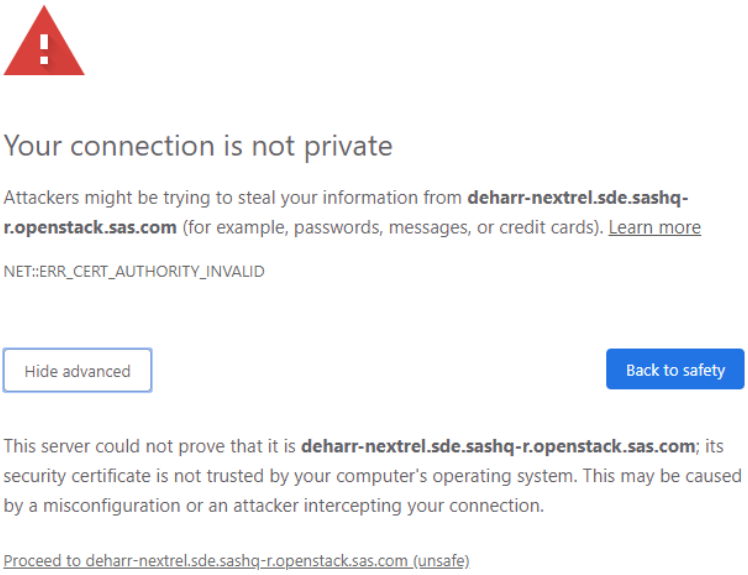
Manufacturers of a majority of web browsers such as Google Chrome, Mozilla Firefox, and Microsoft Internet Explorer deliver a bundle of selected CA certificates along with their products. The installation process configures the browsers to use this bundle of CA certificates by default. This set of CAs is provided by:

- For profit businesses such as DigiCert and Verisign
- Not-for-profit business such as Mozilla
- Governments and NGOs
- Non-profits; notable examples are CAcert and Let's Encrypt. These rely on automated systems to validate claims that an organization owns a web address.

When you use one of these browsers to establish a secure connection to website, the browser follows the TLS protocol and is presented the website's public key in an x.509 certificate. The browser traverses the subject/issuer chain of trust until it gets to the root CA. If this root CA is included in the bundle of trusted certificates that were packaged with the browser software, the browser establishes a secure connection without delay.

Conversely, if the root CA is not included in this bundle of trusted certificates, then you will be presented with a message similar to the one below:

[10] (Certificate authority, n.d.)



You should understand that the technical security around this connection is the same regardless of who issued the certificate being used to provide the public key. The same benefits of AES encryption, Diffie-Helman key exchange, RSA and TLS are in place no matter the provenance of the root CA.

A "public CA cert" is simply used to describe a CA cert that is widely distributed by big software organizations such as Microsoft, Google, Apple, Oracle, Red Hat, etc. These CA certs provide no more security than a self-signed certificate from the standpoint of encryption.

The benefit that is provided by the bundle of CA certificates included by software vendors with their browsers is that the organizations that hold the private keys behind the certificate authorities have professed to exercise some degree of due diligence prior to signing any certificates with their keys.

What this means is practice is that the DigiCert ,Verisign and other organizations that are in the business of signing certificates, take care to ensure that the entities that are asking for signatures really are who they claim to be. The browser vendors: Microsoft, Google, Mozilla believe them enough to include them as CAs thus preventing the warning message above from being displayed.

The digital chain of trust has been explained. The physical chain of trust is

1. User of the browser (you) trusts the vendor of said browser (Microsoft).
2. The vendor of said browser (Microsoft) trusts the organization that provided the root CA (DigiCert).
3. The organization that provided the root CA (DigiCert) trusts the organization that requested the server certificate (amazon.com)

THE MECHANICS OF ENFORCING TRUST

The software you are using is configured to use specific trust store that probably contains the usual public CA certificates. If a person has file system permissions to add certificates to this trust store, then this person can add any CA they like. The mechanics of enforcing trust for most applications come down to file system permissions or repository permissions.

Suppose your company sets up a web server that is accessed by employees. The IT admin can generate a private key and a CA and use this to create his own identity certificate for the server. If this IT admin keeps the private key secret, this CA can safely be added to the trust store on machines controlled by your company. The IT admin adds this CA certificate to all the trust stores on all machines at the company and this web server will now be trusted.

Your company does not need for DigiCert to tell it that it can trust this website because your company controls the website. In fact, if you wanted to use DigiCert, you would simply go to their website and fill in a questionnaire about your company. They would call you back at the number you provided to verify this information and then issue you an identity certificate for the server with the information you provided for a fee. You would then configure the web server to use this certificate. You would save the step of having to add the CA to the trust store because the DigiCert is already there.

So why use DigiCert? Because if you are amazon.com, you do not own all the machines that will be used to connect to your web servers, so you have to pay DigiCert (or another company that has their CA ubiquitously included on all machines in the world) so that your server's identity certificate will be trusted on everyone's machine.

For a self-signed certificate to be trusted, that exact certificate would need to be in the trust bundle on both machines. This is ideal if you have Write permissions to the trust stores on both machines and you keep the private key secret.

If you receive a certificate from an individual or organization that you trust (For example, your employer or a software vendor such as SAS) it is perfectly reasonable to add this CA to the bundle that your browser is configured to use.

Vocabulary

Data at rest: Data at rest is used to describe data in files on media. This can be tape drives, USB memory drives, magnetic disks, either on your local machine or in the cloud.

Data in Motion: Data as it is moving through the network. This can be in the form of electrons moving through a copper ethernet wire, photons moving through a fiber or in the circuits/memory of networking equipment such as switches, routers or your network interface card.

Data in Use: Data as it is being processed by an application.

PKI: The framework and services that provide for the generation, production, distribution, control, accounting, and destruction of public key certificates. Components include the personnel, policies, processes, server platforms, software, and workstations used for the purpose of administering certificates and public-private key pairs, including the ability to issue, maintain, recover, and revoke public key certificates.

HOW SAS USES CRYPTOGRAPHY TO PROTECT YOUR DATA

OVERVIEW

SAS Viya configures all products to encrypt all network traffic by default during the deployment process. All communication between SAS Viya services happens securely using TLS.

SAS uses Ansible and SSH to deploy the ordered products onto the specified machines.

SAS creates its own PKI using the SAS Secrets Manager, an application that creates a private key and uses it to securely issue x.509 certificates.

SAS generates a SAS specific CA and distributes it to all machines that will need to trust sas servers

SAS issues SAS specific identity certificates to each of the servers that accept connections.

SECURE SHELL

If you need to log in to a remote machine to do work, you can use the same security techniques described above to encrypt operating system commands and send them securely over the internet to the remote machine to be decrypted and executed. Almost anything that can be done on a local machine at a command prompt, can be done on a remote machine via secure shell (SSH).

SSH creates a connection between two machines using TLS, the local machine that you are currently logged in to and is the machine from which you initiate the SSH connection. The target machine that is where you intend to run commands remotely.

So you need to log into the remote machine but instead of providing a username and password to establish the connection, you use an RSA keypair authenticate to the target machine.

First you generate the key pair. Then you store a private key in a file on in your home directory the local machine. You store the public key in an authorized keys file in your home directory on the target machine. Then you initiate the SSH session with the command "ssh myusername@hostname". SSH retrieves the public key from the specified user's home directory on the target machine. If the public key corresponds to the private key on the local machine, then you are logged in as the supplied user. Once logged in, you can enter commands on the terminal on your local machine and they will be executed on the target machine.

ANSIBLE

SAS uses Ansible to deploy SAS Viya. Ansible is an automation tool automates the process of running a complex sequence of commands on multiple machines. Ansible uses a file that contains information about the hosts on which to deploy software (inventory.ini), and the set of commands necessary to perform all the task required to deploy the software in files called playbooks.

These include yum and RPM commands, commands to configure files, create/start/stop services, configure the operating system, etc.

All this is done from a single machine called the Ansible controller. Ansible uses SSH to execute the commands necessary to deploy software on the remote, target machines listed in the inventory file.

THE SAS CONFIGURATION SERVER

The SAS Configuration Server is used to reliably store data in files on disk. The server can run as a single process on a single machine, but this would be a single point of failure. SAS recommends that you deploy a minimum of 3 SAS Configuration Servers running on 3 individual machines that can communicate with each other over a network. When deployed in this manner, they form a cluster that is highly available, fault tolerant and consistent data store. This means that even if a subset of the machines on which a SAS Configuration Server is deployed fail or become unreachable due to a network failure, the data remains available and data integrity is assured.

THE SAS SECRETS MANAGER

The SAS Secrets Manager is the backbone of the SAS Public Key Infrastructure. It is the SAS provided certificate authority. The trust you place in the SAS company is represented by the objects that are created by and stored in the SAS Secrets Manager. Once configured, it creates all the ingredients necessary to perform the task of a PKI including:

- Generates and securely stores a private key. There is no mechanism for obtaining this key.
- Uses the private key to create a certificate authority certificate that will serve as the root of trust. This is the certificate that will be included in the store of trusted certificates to establish trust.
- Generates an intermediate certificate and identity certificates that are chained with the CA Certificate.
- Certificate Management

It then functions as a certificate bakery, generating and issuing certificates to all the SAS Viya servers that need them to secure network connections.

The SAS Secrets Manager issues certificates on demand but to authorized entities. To get a certificate, you must prove you are authorized to do so by presenting a valid token. This ensures that the only services authorized by SAS will receive certificates signed by the SAS Viya Intermediate CA and be trusted by other SAS Software.

The SAS Secrets Manager is based on HashiCorp® Vault®, an open source software product.

AUTHORIZATION MANAGEMENT

The SAS Secrets Manager controls identity and authorization by using tokens. During the deployment process, tokens are granted to authorized SAS products with specific policies and permissions.

DATA STORAGE

The SAS Secrets Manager does not create any disk files to store the key or certificate information that it generates. It stores all data in the SAS Configuration Manager®. After all, the SAS Configuration Server is specifically constructed to reliably store information on disk. However, the SAS Configuration Server does not encrypt data before storing it. You cannot have private keys stored on disk in ways that can be discovered. To prevent this from happening, the SAS Secrets Manager application encrypts all information before sending it to the SAS Configuration Server (via a secure network connection) for storage on disk. As a result, the data stored by the SAS Configuration Server is completely useless without the SAS Secrets Manager to decrypt it.

SAS PUBLIC KEY INFRASTRUCTURE SECURE BOOTSTRAP WORKFLOW

There is a dilemma in configuring the SAS Configuration Server and SAS Secrets Manager. The SAS Secrets Manager needs the SAS Configuration Server to store data before it can function. The SAS Configuration Server must be secured with a server identity certificate before the SAS Secrets Manager will trust it to store data. You can't get a certificate from the SAS Secrets Manager until it's up and running. You can see that this creates a circular dependency problem. To get around this dependency problem, the playbook executes an intricate sequence of steps to securely bootstrap the Configuration Server and the Secrets Manager. There is an interval of time where the SAS Configuration Server is secured with

self-signed certificates, which are cumbersome to manage. In the end, the SAS Configuration Manager will be secured using the same CA as all other SAS Viya services.

Here is a high-level overview of the workflow that securely bootstraps the SAS Configuration Server and SAS Secrets Manager:

Actor	Action
Ansible	Execute a script to create self-signed server certificate
Script from prior step (running simultaneously on each machine in the configuration server cluster)	Create self-signed server certificate for the configuration server on given machine Set up SAS Configuration Server to use this certificate and the SAS trust store Use SCP to copy the self-signed certificate (a CA) from this machine to all other SAS Configuration Server machines
Ansible	Execute sas-merge-certificates.sh on each machine that will run a SAS Configuration Server
Sas-merge-certificates script	Add all self-signed certificates (CAs) to the SAS trust store on the local machine
Ansible	Execute SAS Configuration Server start script
SAS Configuration Server start script (running simultaneously on each machine in configuration server cluster)	Finish the SAS Configuration Server setup by creating Access Control Lists and permission settings. Launch a process that blocks until SAS Secrets Manager is available. This process will eventually trigger replacement of self-signed certificates currently in use Start SAS Configuration Server process
Ansible	Execute script to configure SAS Secrets Manager
Script from above step (running simultaneously on each machine in the configuration server cluster)	Set up SAS Secrets Manager to reject any network traffic since it has not yet been fully secured
Ansible	Execute SAS Secrets Manager start script
SAS Secrets Manager start script (running simultaneously on each machine in SAS Secrets Manager server cluster)	Verify that SAS Configuration Server is operational Start SAS Secrets Manager
Ansible	Execute vault-setup script
Vault-setup script (running simultaneously on each machine in SAS Secrets Manager server cluster)	Initialize SAS Secrets Manager Wait for SAS Secrets Manager processes to form a cluster and elect a leader Unseal SAS Secrets Manager. This produces an unseal key, which is a secret that controls access to the server.
Vault-setup script (running on **single** SAS Secrets	Configure SAS Secrets Manager token backend

Manager machine that was elected **leader** in prior step)	Create PKI backend for Root CA, Intermediate CA Create Access Control Roles to manage certificates Use SSH to copy the unseal key, root token, and SAS CA certificate to other SAS Secrets Manager machines Add SAS CA certificate to trust store Generate leaf certificate to secure SAS Secrets Manager Update SAS Secrets Manager config to listen on external networks with above certificate Restart SAS Secrets Manager to implement changes
Ansible	Execute sync script on all machines with SAS Secrets Manager
Wait_until_consul_killed (running simultaneously on each machine in SAS Secrets Manager server cluster)	Block until all SAS Configuration Server certificates on all machines in the SAS Configuration Server cluster have received their server certificates from the SAS Secrets Manager, then reconfigure SAS Configuration Server to use newly minted certificate that is signed with the SAS CA
SAS Configuration Server background script	Request a SAS CA signed certificate from SAS Secrets Manager. Block until every SAS Configuration Server has a SAS CA signed certificate Stop SAS Configuration Server
Ansible	Add SAS CA to the SAS trust store on every machine in the deployment
Ansible	Configure the remaining SAS applications and services

Table 1. Process to Configure the SAS Configuration Server and SAS Secrets Manager

Configuring the SAS Configuration Servers

THE DEPLOYMENT WORKFLOW IN DEPTH

You will now learn how we initiate trust. Trust is very carefully preserved during the entire deployment process. At no time is any operation performed that might be compromised by unsecured network communication.

The chain of trust originates with the system administrator prior to beginning the SAS deployment process. This individual has operating system credentials and permissions that allow him to place SSH keys on the file system. Once the system administrator sets up the SSH keys, he launches the Ansible deployment playbook provided by SAS.

The trust baton is then passed to Ansible which is provided by the trusted operating system vendors and is carefully monitored by the open-source community.

Ansible reads the inventory file and identifies each of the machines (typically 3) that are targeted to run a SAS Configuration server. Ansible then uses SSH to connect to each of

these machines and launches a process that will deploy SAS Viya. SSH connections leverage the trusted SSH keys to secure the connection.

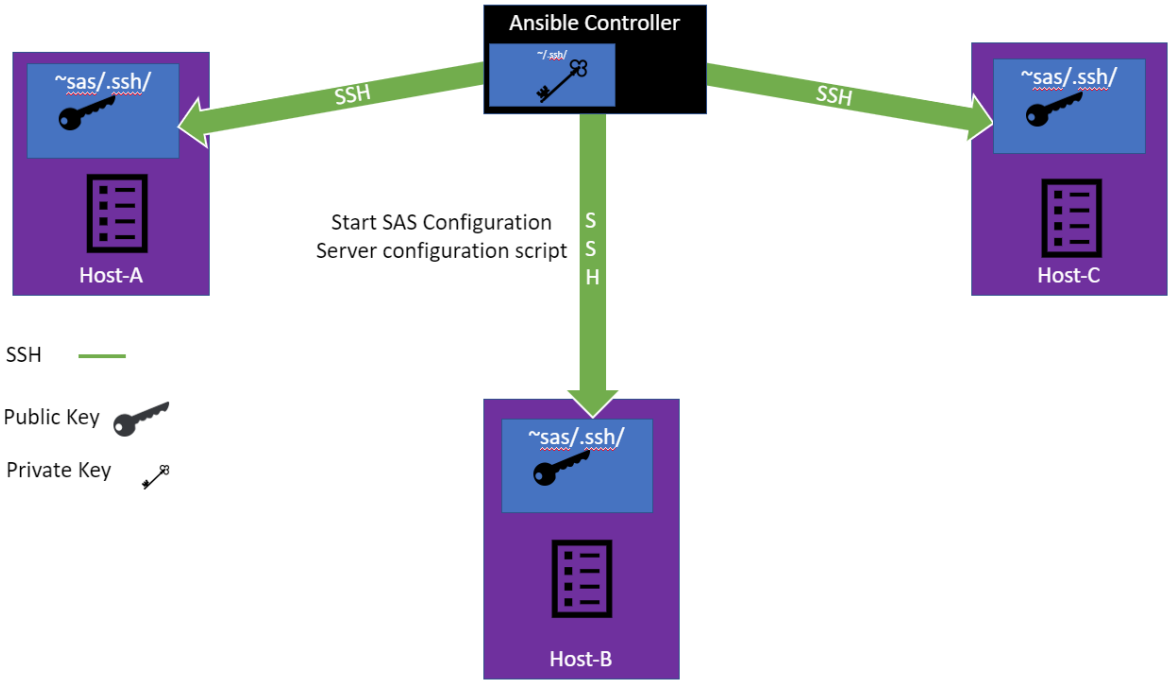


Figure 3. Ansible Using SSH to Connect to SAS Configuration Server Machines and Executing a Configuration Script

The configuration process uses software authored by SAS to perform the next steps. The trust derives from your relationship with SAS. SAS rigorously enforces and audits secure software development and delivery practices so that your trust in SAS software is well placed.

SAS packages this software into RPMs that are digitally signed by SAS before being placed in a repository on the internet. The SAS RPM files are installed using the root login and operating system capabilities that validate the SAS signatures in the RPMs before installing the software.

The baton of trust is now passed via SSH to the SAS Configuration Server deployment script on each of the targeted machines. These scripts create a self-signed certificate and private key on each of the machines. Now we have 3 self-signed certificate and key pairs on 3 different machines.

The next step is to configure the SAS trust stores so that the 3 SAS configuration servers will trust one another. This will require the trust store on each of the 3 machines to contain certificates from the 2 other machines in the cluster. SSH is used to install the self-signed

certificate into the SAS trust store the other machines.

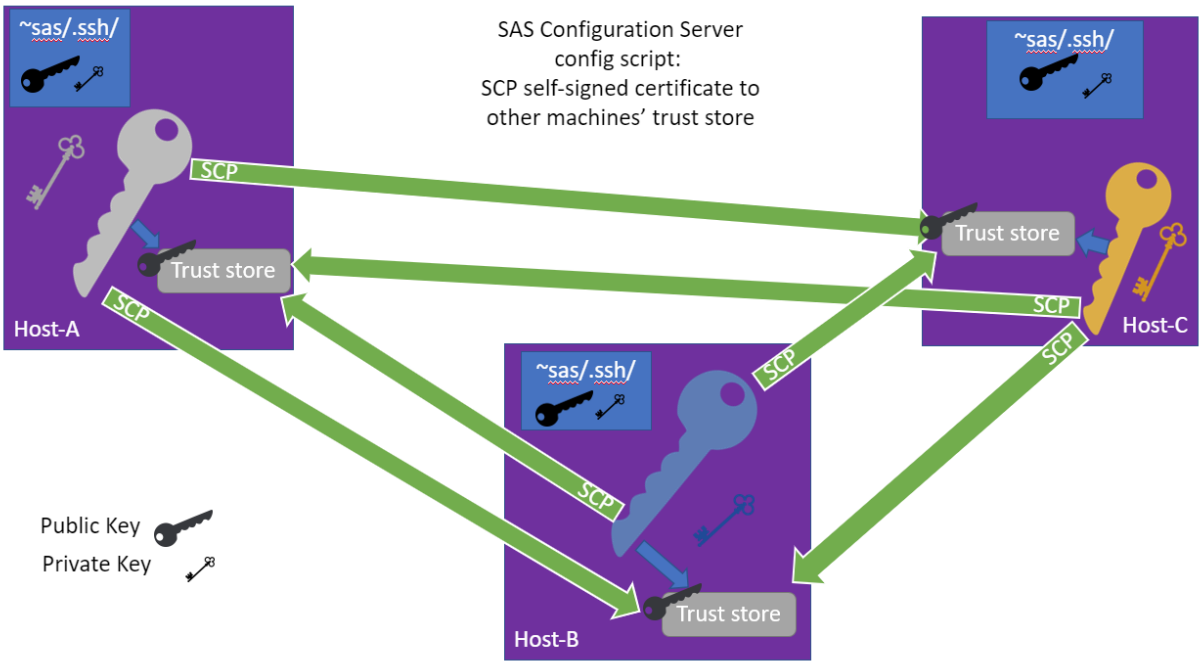


Figure 4 SAS Configuration Server Configuration Script uses SCP to Copy Certificates to the Trust store on Other Machines Running the SAS Configuration Server

Once the SAS trust store on every machine contains CAs from each of the other machines, the deployment script starts the SAS Configuration Server process. The servers can now use TLS communicate with each other securely. They form a cluster and elect a leader. The leader finishes the configuration process and the SAS Secrets Manager PKI becomes functional.

At this point the servers are using self-signed certificates for security. To facilitate the secret renewal, they will be swapped for certificates signed by the SAS Viya Intermediate CA once the SAS Secrets Manager is configured. To do this, we kick off a background process that waits for the SAS Secrets Manager to be configured and secured.

Configuring the SAS Secrets Manager

Once the SAS Configuration server is running, control returns to Ansible. Ansible then begins to process the SAS Secrets manager playbook. The process is very similar to the SAS Configuration Server. Ansible runs a script runs to configure and start a SAS Secrets Manager service on the same set of machines that are running SAS Configuration Servers.

Because the SAS Secrets manager does not yet have any certificates with which to secure communications, it will refuse all network connection requests. The SAS Secrets Manager uses inter-process communication to work with the local SAS Configuration Server and elect a leader. Once a leader is selected, that instance will create the first durable secrets that will secure your SAS deployment. These are the unseal key and the initial root token. These unseal key is very important as this is used to encrypt and decrypt the information stored by the SAS Secrets Manager. It is stored in a file with secure permissions. It must be kept secret and it must be safely maintained. If this unseal key is lost, you will not be able to

recover the secrets stored in the SAS Secrets Manager and your entire SAS Deployment will become unfunctional.

Once the leader SAS Secrets Manager is unsealed, a private key is generated that can be used to digitally sign certificates. With this, the SAS Secrets Manager becomes a certificate authority and a PKI. Certificates used to secure SAS network communications and data in motion will be issued by this SAS certificate authority. At this point the root CA certificate file is created. This CA Certificate has a subject that lists "SAS VIYA Root CA" as its name. It is added into the SAS trust store on this SAS Secrets Manager machine that was elected the leader. The machines that lost the leader election have been anxiously awaiting this moment. Once the CA Certificate file is generated, the non-leader machines use SSH to copy the Root CA from the leader node and include it in their trust stores.

At this point the SAS PKI infrastructure is fully functional and the SAS CA has been included in the SAS trust stores. The self-signed certificates that were generated to bootstrap the SAS Configuration Servers are replaced by certificates issued by the SAS PKI and the servers are restarted using these certificates in their normal mode of operation.

SAS Configuration Server Receives Certificate Signed by SAS Viya Intermediate CA

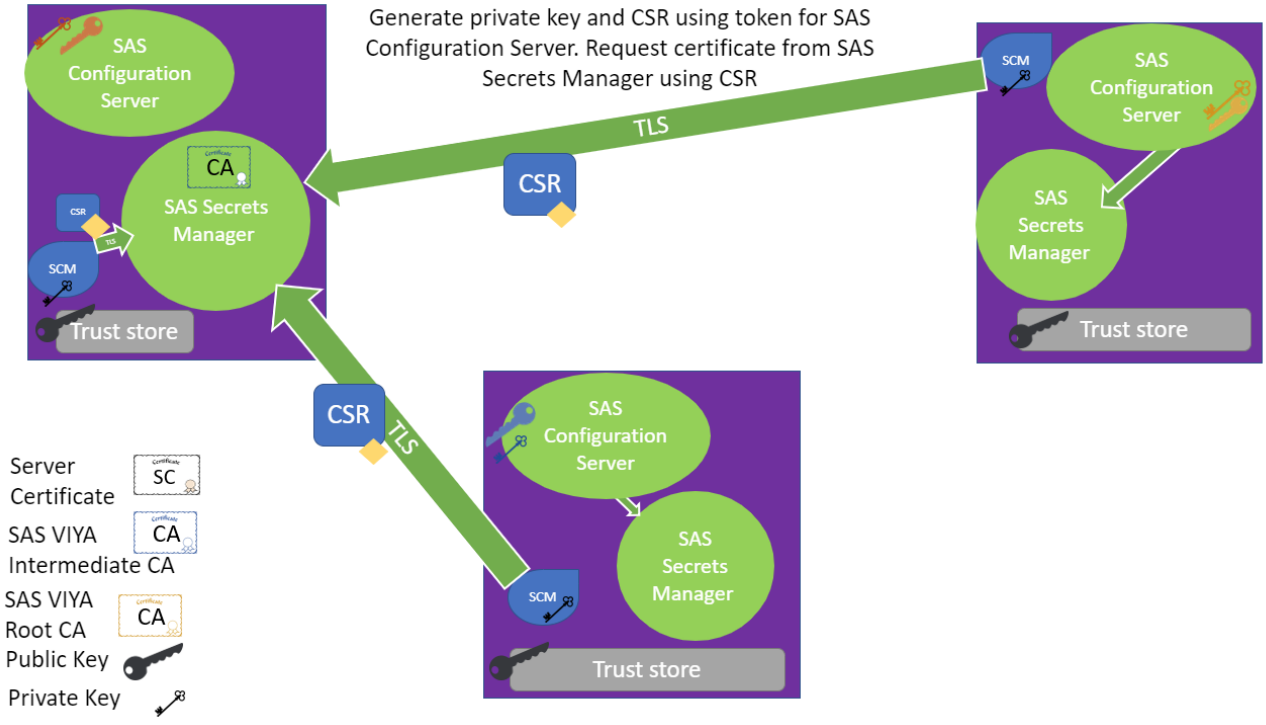


Figure 5. Internal Tool Sending CSR to SAS Secrets Manager to be Signed

HOW SAS USES THE INFRASTRUCTURE TO SECURE SERVICES

At this point in the deployment, the SAS Configuration Server and the SAS Secrets Manager are secured using TLS certificates issued by the SAS Viya Root CA and generated by the SAS PKI. The SAS Secrets Manager is accessible securely over the network.

Now, Ansible will run the playbooks to deploy all the remaining SAS products on all the other machines listed in the inventory file. This will include adding the SAS Viya Root CA to the SAS trust store.

All SAS applications and services that come after the SAS Configuration Server and SAS Secrets Manager follow the same pattern to secure their network traffic. Ansible uses SSH to securely provide the individual service or application with authentication token for use with the SAS Secrets Manager.

This token is used each time the SAS service or application starts to get a new identity certificate specifically for the individual service or application that is issued by the SAS Viya Root CA.

This completes the process. Now every single SAS authorized service or application will get a brand-new identity certificate each time it starts and this certificate will be trusted by all other SAS applications and services.

Now any service or client connecting to any service in a SAS Viya deployment can be sure that any information being sent is encrypted and that the service is trustworthy.

MANAGING THE TRUST STORE – WHO CAN YOU TRUST?

The list (or bundle) of CA certificates that are trusted by applications is called the trust store or trust bundle. As noted earlier, clients will not complete TLS connections to servers unless the identity certificate returned by the server is issued by a CA for which the root CA certificate is included in the trust store. We also discussed how vendors of browsers and operating systems included a trust store with their software.

To maintain isolation of the OS and browser trust stores that are managed by the IT department, SAS provides its own trust store that is used by SAS applications.

The SAS trust store is based on the Network Security Services (NSS) Root Certificate Store. The NSS store is a collection of trusted root certificates that is maintained by Mozilla and is included in Firefox's trust stores and many other trust stores. This trust store contains all the major commercial CAs and is largely equivalent to the trust stores provided by the major OS and browser vendors.

If you need for the SAS programs to trust websites in your organization's infrastructure that are secured by a certificate that is not in the NSS trust store, you should follow the instructions in the SAS Viya Administration Guide for adding additional certificates to the SAS trust store.

The Apache httpd server is the one SAS configured application that is not secured with a certificate issued by the SAS Viya Root CA. This is because that product is the "front door" to the SAS web applications. Since the web browsers that will be connecting to this web server (Chrome, IE, Safari, Mozilla Firefox, etc) are configured to use the operating system trust store and not the SAS trust store, they would not trust the SAS Viya Root CA. The Apache installation script creates a self-signed certificate by default so that the server will work out of the box, but this certificate is not added to the OS trust store. This means the Apache httpd server will trigger warning messages from browsers. This does not mean that the network traffic is not secured. It is secured. The message means that the certificate is not in the trust store. SAS recommends that you replace this certificate with whatever certificates conform to your IT standards for web servers.

As the SAS Viya Root CA created for your deployment is globally uniquely, it will not be present in your system trust store. This means that applications not provided by SAS (such as browsers), will not trust the SAS Viya Root CA. If you use a browser to connect to a SAS

server such as CAS directly. This too will cause it to issue a warning that the certificate is not in the operating system trust store. However, you should not have a need to connect directly to SAS servers, using the Apache web server as a proxy, will eliminate this issue.

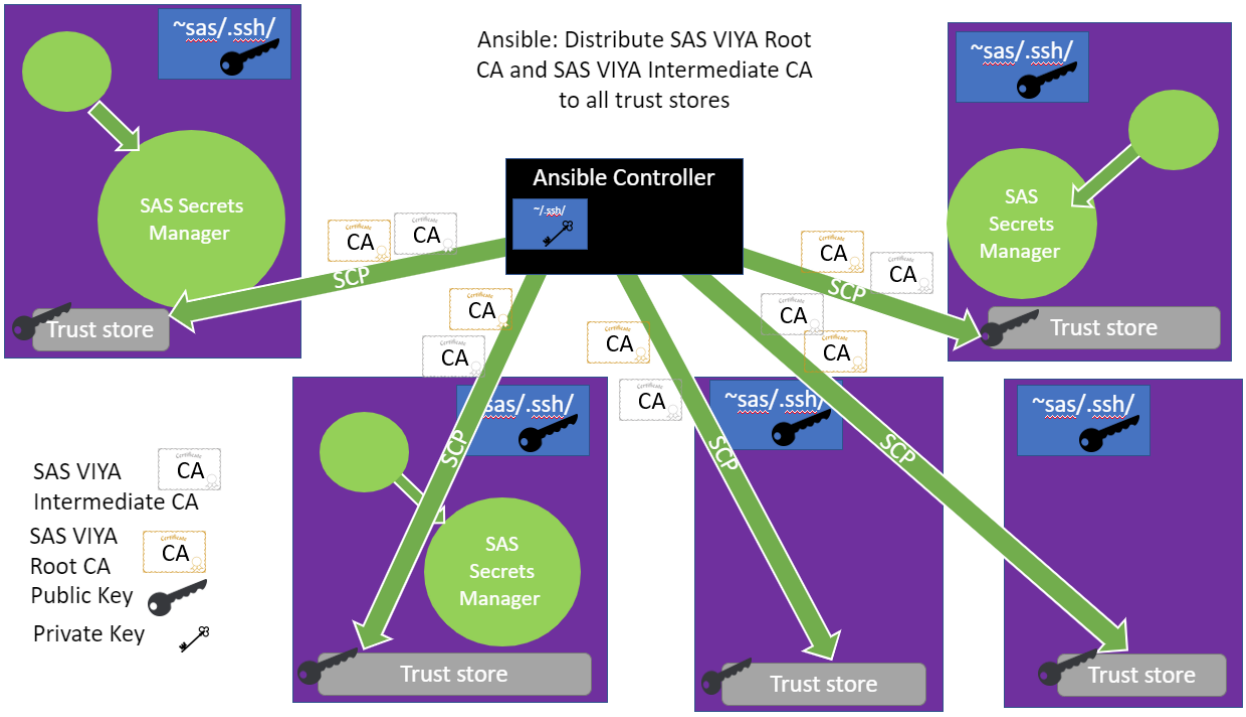


Figure 6. Diagram of Ansible Using SCP to Copy Certificates Around and Rebuild the Trust store on all Machines in the Deployment

HOW SAS ROTATES SECRETS?

As you know, secrets can leak out of organizations. Employees can leave a company, consultants can gain access to them while performing IT tasks, etc. Therefore, it is important to rotate secrets periodically and when leakage is suspected. The certificates and private keys used to secure SAS network traffic are no different. They must be rotated periodically. As described earlier, SAS automates this process by generating a new private key and certificate every time a service starts.

The only services that do not follow this paradigm are the SAS Configuration Server and SAS Secrets Manager due to their circular dependence. SAS provides a utility play that renews all secrets used for securing network traffic. The renew-security-artifacts.yml ships with the SAS deployment playbook. When run, it updates:

- certificates used to secure and the SAS Configuration Server
- certificates used to secure the SAS Secrets Manager
- other secrets used by the SAS Secrets Manager
    - unseal key,
    - root token,
    - individual service tokens

Additionally, the SAS trust store on all machines is regenerated to remove the old SAS Viya Root and Intermediate CAs and replace them with the newly generated SAS Viya Root and Intermediate CAs.

CONCLUSION

The rapidly changing uses and wide availability of the internet has greatly improved the ability to communicate with people and businesses around the world. There are many transactions that you will have online that you would like to be confident that only the entity you are trying to interact with can read your message. Encryption and X.509 certificates provide you those assurances if private keys are safely kept. You can feel confident that your SAS Viya deployment is securely deployed, configured, and administered because the entire deployment is completed using SSH and SCP to securely deploy and configure the environment, and all services in the ecosystem use short-lived certificates to communicate. These certificates are signed by a central, well protected certificate authority that uses token-based authentication to verify the requestor's identity.

REFERENCES

Advanced Encryption Standard. (n.d.). Retrieved March 8, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Certificate authority. (n.d.). Retrieved March 12, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Certificate_authority

Deepfake. (n.d.). Retrieved March 11, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Deepfake

Diffie Hellman key exchange. (n.d.). Retrieved March 13, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

Digital signature. (n.d.). Retrieved March 13, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Digital_signature

HashiCorp. (n.d.). *Consensus Protocol*. Retrieved from Consul by HashiCorp: https://www.consul.io/docs/internals/consensus.html

HashiCorp. (n.d.). *Seal/Unseal*. Retrieved from Vault by HashiCorp: https://www.vaultproject.io/docs/concepts/seal.html

HashiCorp. (n.d.). *Vault Curriculum*. Retrieved March 11, 2019, from HashiCorp Learn: https://learn.hashicorp.com/vault/

History of Cryptography. (n.d.). Retrieved March 11, 2019, from Wikipedia: https://en.wikipedia.org/wiki/History_of_cryptography

Kerckhoffs's Principle. (n.d.). Retrieved March 11, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Kerckhoffs%27s_principle

RSA (cryptosystem). (n.d.). Retrieved March 11, 2019, from Wikipedia: https://en.wikipedia.org/wiki/RSA_(cryptosystem)

SAS Institute Inc. (2018). *SAS Viya 3.4 for Linux: Deployment Guide.* Cary, NC: SAS Institute Inc.

Secure Shell. (n.d.). Retrieved March 12, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Secure_Shell

Triple DES. (n.d.). Retrieved March 7, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Triple_DES

Wikipedia, the free Encyclopedia. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/

X.509. (n.d.). Retrieved March 13, 2019, from Wikipedia: https://en.wikipedia.org/wiki/X.509

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Alec Fernandez
SAS Institute
Alec.Fernandez@sas.com

Destiny Harrell
SAS Institute
Destiny.Harrell@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.