EPICS Security Architecture

Whitepaper

Revision 8.0 00 September 2024

George McIntyre (Level N)
Greg White (SLAC)
Michael Davidsaver, Bob Dalesio (Osprey)
Kay Kasemir (ORNL)

Abstract

This document describes analysis and proposed architecture designs to significantly improve the cyber security of the Experimental Physics and Industrial Control System, EPICS. EPICS is the foundational software of the control systems of many large experimental facilities such as particle accelerators, light sources, fusion research tokamaks and other collaborating institutions worldwide.

Typically, such control systems are protected only by a sequestered network; the protocol is unencrypted, authorization is based on usernames which can easily be faked, there is no authentication of Input/Output Controller (IOC) computers (of which there may be hundreds), and no software signing.

The document presents solutions to the challenges uncovered in analysis, plus policy and configuration questions for the consideration of SLAC, the EPICS community, and the U.S. Department of Energy (DOE). For a full list of the key decisions that have been made see `EPICS Security Key Decisions [1]`.

Contents

1. Introduction 9
2. Glossary 10
3. Technology Background 14
    3.1. Zero Trust 14
    3.2. Public and Private key pairs 15
    3.3. How Certificates guarantee authenticity 16
    3.4. Digital Certificates 17
        3.4.1. X.509 Server Certificates 17
        3.4.2. X.509 Client Certificate 18
        3.4.3. Secure PVAccess custom X.509 Certificate Extension 18
    3.5. OCSP 18
    3.6. Network Protocol Encapsulation 18
4. Problem Statement 20
5. Architecture Summary 21
    5.1. Architecture Overview 21
    5.2. High Level Design Topics 22
        5.2.1. Backwards Compatibility 22
        5.2.2. Authentication Methodology 23
        5.2.3. Support for External CAs 23
        5.2.4. Operational impacts 24
    5.3. Summary of Proposed Changes 26
6. Detailed Functional Specifications 27
    6.1. EPICS Agent Environment Variables 27

6.2. PVACMS *Environment Variables* 28

6.3. JWT AD *Environment Variables* 31

6.4. KRB AD *Environment Variables* 32

6.5. LDAP AD *Environment Variables* 32

6.6. Basic Credentials AD *Environment Variables* 33

6.7. OCSP *Environment Variables* 33

6.8. Securing the PVAccess Protocol 34

6.8.1. *TLS Implementation* 34

6.8.2. SEARCH and SEARCH RESPONSE messages 34

6.8.3. URI scheme 35

6.8.4. Ports 35

6.9. Authentication 36

6.9.1. Server and Client Startup 36

6.9.2. Authentication Integration Framework 37

6.9.3. Certificate Creation Steps 37

6.10. Authentication Daemons 37

6.10.1. Authentication Daemon credential verification 38

6.10.2. AUTHZ Message 39

6.10.3. Types of independent verifications supported 40

6.10.4. Authentication method implementation 40

6.11. Operating a Secure PV Access network 44

6.11.1. TLS operating attributes 44

6.11.2. Authentication Mode 45

6.11.3. Operational Exceptions 45

6.11.4. Diskless Server bootup with TLS 46

6.12. Certificate Management Features 47

6.12.1. Certificate Authority 47

6.12.2. Automatically provision certificates when missing or invalid 47

6.12.3. Revoke certificates 48

6.12.4. Create PKCS#12 file and Root Certificate 48

6.12.5. PVACS: Certificate Status over PVAccess 49

6.12.6. PVACS stapling integration with TLS handshake 49

6.12.7. PVACS integration with PVAccess server for own-certificate verification 50

6.13. Notification System 50

6.14. Supporting non-EPICS network clients 50

6.14.1. Add OCSP-PVA service. 50

6.15. Gateways 50

6.16. Tool updates 50

6.16.1. softIocPVX (qsrv) 51

6.16.2. pvacms 51

6.16.3. odbc-pva 51

7. Source, Building, and Releases 52

7.1. Build Macros 52

7.2. Source Control 52

7.3. TLS external libraries 53

7.4. New External Libraries 53

7.5. Add new unit tests 54

7.6. Releases, versions, and roadmap 54

7.6.1. Update PVXS version 54

7.7. Licensing, releases, versions, and roadmap 55

7.7.1. Licenses 55

8. Performance Goals and Testing 57

8.1. Perf goals 57

8.2. Rollout 57

9. Works Cited 59

10. Index 65

List of Figures

Figure 1 Project Goals 9

Figure 2 Certificate State Transitions 12

Figure 3 Zero Trust approach to security..... 14

Figure 4 Perimeter based approach to security. 14

Figure 5 Generating Key Pairs..... 15

Figure 6 X.509 internal format. 17

Figure 7 X.509 encoding and packaging formats. 17

Figure 8 Encapsulation of PVAccess messages within TLS 18

Figure 9 Secure PVAccess Session..... 20

Figure 10 System Components 21

Figure 11 Network resource access in proposed system 22

Figure 12 Basic TLS Protocol Sequence 34

Figure 13 Port usage options for Secure PVAccess..... 35

Figure 14 Server and Client certificate acquisition startup..... 36

Figure 15 Authentication Integration Framework 37

1. Introduction

The project goals were derived from the Executive Office memorandum, `“Toward Zero Trust Cybersecurity Principles” [2]` as interpreted for an EPICS control system of a large experimental facility:

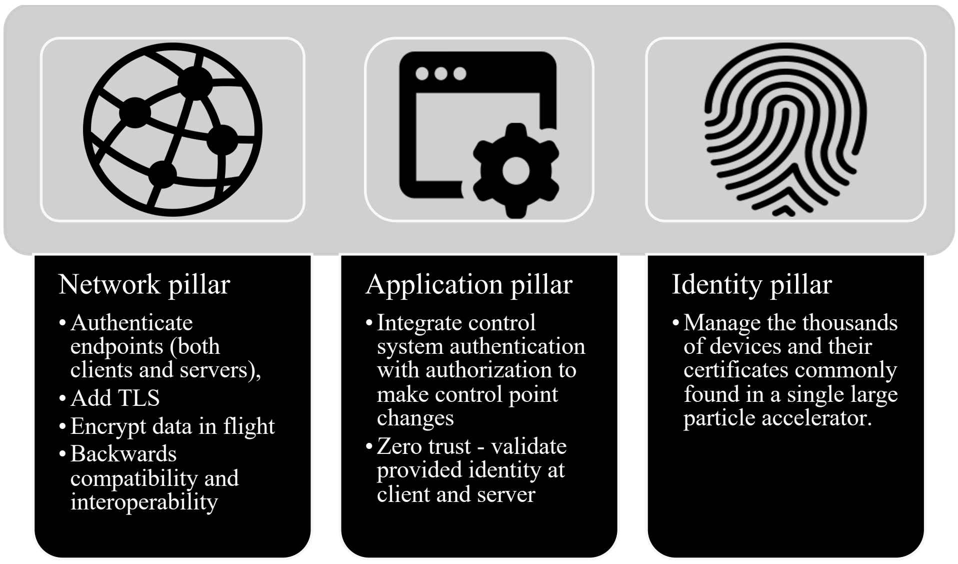

Network pillar
• Authenticate endpoints (both clients and servers),
• Add TLS
• Encrypt data in flight
• Backwards compatibility and interoperability
Application pillar
• Integrate control system authentication with authorization to make control point changes
• Zero trust - validate provided identity at client and server
Identity pillar
• Manage the thousands of devices and their certificates commonly found in a single large particle accelerator.

Figure 1 Project Goals

The work is based on prior analysis of `EPICS cyber security [3]`, analysis of `SLAC accelerator’s cyber vulnerabilities [4]`, and design to add `Transport Layer Security (TLS) to EPICS [5]`.

This is the third milestone delivery to satisfy the FY23/24 SLAC DOE proposal `SLAC Executive Order 14028 Cybersecurity Project – Internal DOE Funding Request [6]`

2. Glossary

- *Auth (Authentication) vs AuthZ (Authorization)*. In cybersecurity, these abbreviations are commonly used to differentiate between two distinct aspects of the security process. *Authentication* refers to the process of verifying the validity of the credentials and claims presented within a security token, ensuring that the entity is who or what it claims to be. *Authorization*, on the other hand, is the process of determining and granting the appropriate access permissions to resources based on the authenticated entity's credentials and associated privileges.
- *Access Security Group (ASG), User Access Group (UAG)*. In EPICS, authorization is managed by using an Access Security configuration file that contains UAGs (lists of users and their groups), and SAGs (definitions of which user groups can access what resources), to control access to resources/devices the EPICS server controls.
- *Authentication Daemon (AD)*. These are new components that we envisage for the Secure PVAccess Network that will work with network clients and server to monitor and update their X509 certificates based on external input. For example, the Kerberos AD will update the X509 certificate using the CCR protocol TYPE 2, message for the PVACMS, in response to new and updated Kerberos tickets – kinit/ticket renewals. The JWT AD will update the X509 certificate using a CCR protocol TYPE 1, message for the PVACMS in response to a client receiving a new JWT or when the JWT needs refreshing.
- *CA – Certificate Authority*. An entity that signs, and issues digital certificates. Each site where EPICS is installed will use the proposed PVACMS as their CA.
- *Certificate's Subject. This is a way of referring to all the fields in the X.509 certificate that identify the entity. These are:- **CN**: common name e.g.* `slac.stanford.edu`*; **O**: organization e.g.* `Stanford National Laboratory`*; **OU**: organizational unit e.g.* `SLAC Certificate Authority`*; **C**: country e.g.* `US`*.*

  *In Secure PVAccess we will use the **CN** field to store the device name e.g.* `KLYS:LI16:21`*, or username e.g.* `greg`*, or process name e.g.* `archiver`*. We will use the **O** field to store the hostname e.g.* `centos01`*, IP Address e.g.* `192.168.3.2`*, realm e.g.* `SLAC.STANFORD.EDU`*, or another domain identifier.*
- *Certificate Creation Request, CCR*. In Secure PVAccess, we will use this term to describe a message that will be sent to the PVACMS to request a new certificate. This message will be sent by any EPICS agent that needs to establish an authenticated secure connection with a peer over Secure PVAccess. It will contain the subject of the requested certificate, the validity date range, whether the certificate is for a client or server, the public key of the requested certificate, and any verification information such as message signatures used by supported authentication methods.
- *Client Certificate , Server Certificate, X.509*. In cryptography, a client certificate is a type of digital certificate that is used by client systems to make authenticated requests to a remote server which itself has a server certificate. They contain claims that are signed by a CA that is trusted by the users (client and server) of the certificates. For this work we will only be working with X.509 certificates.
- *Custom Extension, for X.509 Certificates*. X.509 allows for the inclusion of custom extensions, (`RFC 5208 [7]`), which are data blobs encoded within certificates and signed alongside other certificate claims. In our implementation, we use a custom extension to define how certificate validity is established. Specifically, the extension mandates that a certificate shall only be considered valid if its status is successfully verified against the PVACMS provided within the extension.
- *Data in flight, Data at rest, Data in use*. Terms usually applied to places we may wish to encrypt data. *Data at rest* is safely stored on an internal or external storage device. *Data in flight*, also known as data in motion, is data that is being transferred between locations over a private network or the Internet. *Data in use* is an information technology term referring to active data which is stored in a non-persistent digital state typically in computer random-access memory (RAM), CPU caches, or CPU registers.
- *Default Credentials AD.* A simple Authentication Daemon that is provided by default, that will get the name of the *logged-in user* and determine the *hostname* (or the *process name* or the *IP address*) and then use these default credentials to request a certificate from PVACMS in a CCR. This certificate is installed for use by the EPICS agent which will automatically reconfigure connections to use it if it is configured for Secure PVAccess communications. Such certificates are always generated in the state of `PENDING_APPROVAL` so they will only be valid if approved by a network administrator.
- *Diskless Server, Diskless Node, Network Computer, Hybrid Client*. A network device without disk drives, which employs network booting to load its operating system from a server, and network mounted drives for storage.

- *Endpoint*. Physical devices that connect to and exchange information with a computer network.
- *EPICS – Experimental Physics and Industrial Control System*. A large software framework for distributed control systems of large experimental instruments such as particle accelerators and radio telescopes. The target of this work.
- *EPICS Agents*. This is a documentation shortcut to refer to EPICS clients, servers, gateways, and tools.
- *EPICS Security*. The existing EPICS technology for user Authorization (it does not include Authentication).
- *IOC – Input/Output Controller*. Provides direct control and input/output interfaces for each accelerator subsystem. The standard crate uses either the VME or VXI standards, a Motorola 68040 processor, and network communications. PVAccess Servers compile to run natively on IOCs of all types including SoftIOCs (software IOCs) with device control interfaces.
- *IOC Shell*. An interactive shell that is available for a PVAccess Server that accepts a set of commands with optional parameters that can configure the Server and/or access data that it provides.
- *IP Protocol – Internet Protocol*. A standard set of requirements for addressing and routing data on the Internet.
- *JWT – JSON Web Token*. (`RFC 7519 [8]`) A compact URL-safe means of representing claims to be transferred between two parties. The token is signed to certify its authenticity. It will generally contain a claim as to the identity of the bearer (**sub**) as well as validity date ranges (**nbf**, **exp**). The JWT AD will package the token into a CCR message and the PVACMS will, using the compiled-in TYPE 1 JWT verifier, verify the signature of the token with the token issuer that it trusts, and then, because it will then be able to trust all the claims in the token, it will create a certificate that contains all those trusted claims to return to the JWT AD in exchange. The JWT AD will install the certificate for use by an EPICS agent.
- *JWT AD – JWT Authentication Daemon*. Runs on alongside an EPICS agent and provides an HTTP service to receive JWT tokens. When a new or updated token is sent to this service, it makes a CCR request to PVACMS to create or update an X.509 certificate for the EPICS agent. PVACMS verifies the token using a built-in TYPE 1 verifier before issuing the certificate, which is generated in the VALID state. A web page can easily interact with the JWT AD by sending an HTTP POST request with the token whenever it is created or renewed, ensuring certificates are always up to date with the latest token data.
- *Kerberos, Kerberos Ticket*. Protocol for authenticating service requests between trusted hosts across an untrusted network, such as the internet. Kerberos support is built into all major computer operating systems, including Microsoft Windows, Apple macOS, FreeBSD and Linux. A Kerberos ticket is a certificate issued by an authentication server (Key Distribution Center - KDC) and encrypted using that server's key. Two ticket types: A *Ticket Granting Ticket* (TGT) allows clients to subsequently request *Service Tickets* which are then passed to servers as the client's credentials. An important distinction with Kerberos is that it uses a symmetric key system where the same key used to encode data is used to decode it therefore that key is never shared and so only the KDC can verify a Kerberos ticket that it has issued – clients or servers can't independently verify that a ticket is valid. An EPICS agent needing to get a certificate will need to contact PVACMS using GSSAPI to be authenticated.
- *Kerberos AD - Kerberos Authentication Daemon*. This daemon can be executed on any client/server machine and is responsible for monitoring a configurable Kerberos identity for the availability of tickets. When tickets become available or are renewed, the daemon automatically generates CCRs (Certificate Creation Requests) to PVACMS in order for it to create or update the appropriate X.509 certificate for use by an EPICS agent. This AD will use Kerberos APIs to communicate with the KDC (via GSSAPI) to generate an appropriate CCR message. PVACMS will use a compiled-in TYPE 2 verifier to verify the Kerberos-specific CCR data with data provided by the KDC (a `keytab` file) before accepting the claims and generating the certificate. Certificates created by this AD are generated in the `VALID` state.
- *OAuth – Open Authorization*. An open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites or online services but without giving them the passwords. In the case of Secure PVAccess a client will be able to be authenticated by OAuth and have the credentials bundled into a JWT token, then passed to the *JWT AD* which will use it to obtain a new certificate which it will install for an EPICS agent to use to make a connection to a Secure PVAccess server.
- *OCSP – Online Certificate Status Protocol*. A modern alternative to the Certificate Revocation List (CRL) that is used to check whether a digital certificate is valid or has been revoked.
- *Particle Accelerator Control System*. The control system of a particle accelerator is distributed heterogenous computing system to sense the status of the various devices used to generate, accelerate, guide and diagnose the beam in an accelerator. Control systems can be composed of thousands of participating endpoint hardware and software. EPICS is a framework to build the control and timing systems which coordinate these devices, and their user and scientific computing interfaces. The goal of this

work is to ensure that the control system can only be accessed by authenticated users based on the authorization rules stored in the EPICS Security authorization system.

- *PKCS#12 – Public Key Cryptography Standard 12*. In cryptography, PKCS#12 defines an archive file format for storing many cryptography objects as a single file. It is commonly used to bundle a private key with its X.509 certificate and/or to bundle all the members of a chain of trust. It is defined in `RFC 7292 [9]`
- *PV – Process Variable*. Data entities that describe a single aspect of the process or device under control. PVs are the named targets of all PVAccess Search requests and a PVAccess Server that is responsible for access to a PV will respond to such requests.
- *PVA – PVAccess*. A data and control protocol which interconnects end points in an EPICS network. PVAccess allows clients to access any named PV on the network and, with authorization rules, set and read their values. A key element of hardening the security of EPICS, planned here, is to add authentication to PVA communications.
- *PVAccess Server*. The software component acting as the Server role in a PVAccess Protocol exchange. This can be running as a *SoftIOC* which is a software only service that responds to PVAccess requests, or as an embedded server running inside an *IOC*.
- *PVACMS – PVAccess Certificate Management Service*. The proposed clustered service to provide automated provisioning of certificates, management of certificate status lifecycles including *validity, expiration, revocation, approvals* and *rejections*, and a certificate status service all delivered over PVAccess. Clients of this service will include: Secure PVAccess servers, clients, or gateways. The service works over an unsecured PVAccess connection, but its certificate status payloads are packaged as OCSP responses encoded in *PKCS#7 SignedData* structure (`RFC 5652 [10]`).

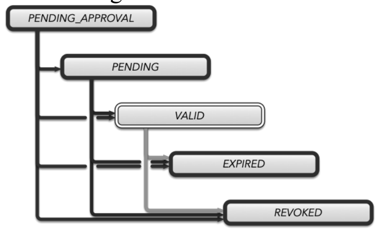


Figure 2 Certificate State Transitions

- *PVACS – PVAccess Certificate Status*. This is the equivalent of OCSP but implemented using PVAccess. It provides timestamped certificate status delivered over the PVAccess Protocol and encapsulated using the *PKCS#7*.
- *PVACS Stapling*. This is the equivalent of OCSP stapling but implemented using PVACS. It will request the server certificate status from PCACMS during TLS handshake and will pin it to the TLS message. Using a callback, on the client side the pinned status is retrieved from the handshake message and decoded to verify the server's certificate status. This speeds up connection times as the client does not need to query the server's certificate status after connection. Note that the client still must verify their own certificate status.
- *Secure PVAccess*. This is the name with which we refer to the final product of this work. More specifically the PVAccess Protocol enhanced by the deliverables of this work – TLS, OCSP, and Certificate Management.
- *Site Authentication Methods*. In Secure PVAccess these are methods of determining the authenticity of credentials. 1) They provide methods of retrieving the credentials via APIs. 2) They provide methods to add method-specific data to CCRs. 3) They optionally provide TYPE 1 or TYPE 2 verification adapters for compilation into PVACMS to authenticate the method-specific data. And finally, 4) they provide methods to update the X.509 certificate used by EPICS agents in a Secure PVAccess network. There are two different types of supported Site Authentication Methods: TYPE 1) with *independently verifiable tokens* – can be verified independently of source, e.g. JWT, or TYPE 2) with *source verifiable tokens* – need source interaction or cooperation to corroborate the claims, e.g. Kerberos.
  - PVACMS handles TYPE 1 auth. methods by checking the signature of the independently verifiable tokens that are added directly to the CCR, by using compiled-in method-specific adapters to make sure that they are signed by a trusted source or calling out to the trusted source to verify the token.
  - PVACMS handles TYPE 2 auth. methods by using method-specific APIs to add verifiable data (e.g. a token equivalent to a Kerberos token) to the CCR message and then, in PVACMS, using method-specific APIs to verify that data by calling out to an external authority (or using information shared by an external authority – e.g. a `keytab` in the case of Kerberos).
  - When no authentication token is available, basic credentials may be provided to the PVACMS. Basic credentials are the logged-in **username**, the **hostname**, the **process name** and/or the **IP-address**. When using basic credentials, verification of the authenticity of the credentials does not take place. All tokens generated by this method must be `APPROVED` by an administrator before they are valid on the network.
- *SKID – Subject Key Identifier*. The subject key identifier identifies the subject of the certificate. In simple terms a certificate is nothing more than a mechanism for certifying that he bearer of the certificate has the private key. So, the SKID is a way of identifying the private key so that if it is used to generate a new certificate the bearer is identified as the same. Its saying "This is

my X" where X can be a process, machine, IOC, service, or anything that can participate in a Secure EPICS network. In practice it simply makes a hash of the public key, as the public key has a one-to-one relationship to the private key. An EPICS agent can keep the private key in as separate PKCS#12 file to the certificate so that it can be used to generate new certificate when the old one expires and will retain the same SKID on the network. You can't generate a new certificate with the same SKID while a prior one has not `EXPIRED` or been `REVOKED`.

- *TCP – Terminal Control Protocol*. A communications standard that enables application programs and computing devices to exchange messages over a network.
- *TLS – Transport Layer Security*. A cryptographic protocol designed to provide communications security over a computer network. The ubiquitous successor to SSL and used to secure HTTP communications (HTTPS).
- *Zero Trust*. Zero Trust security is an IT security model that requires strict identity verification for every person and device trying to access resources on a private network, regardless of whether they are sitting within or outside of the network perimeter.

3. Technology Background

3.1. Zero Trust

Zero Trust is a security framework that requires all users, whether in, or outside, an organization's network, to be authenticated, authorized, and continuously validated for security configuration and posture before being granted, or keeping access to, applications and data. Zero Trust assumes that there is no traditional network edge; networks can be local, in the cloud, or a combination or hybrid with resources anywhere as well as workers in any location. Traditionally, labs maintain a secure network perimeter and permit access only to users that are inside the perimeter. Control device resources are typically implicitly trusted. Control data is not presently encrypted in EPICS. These are all counter to the Zero Trust security principle.

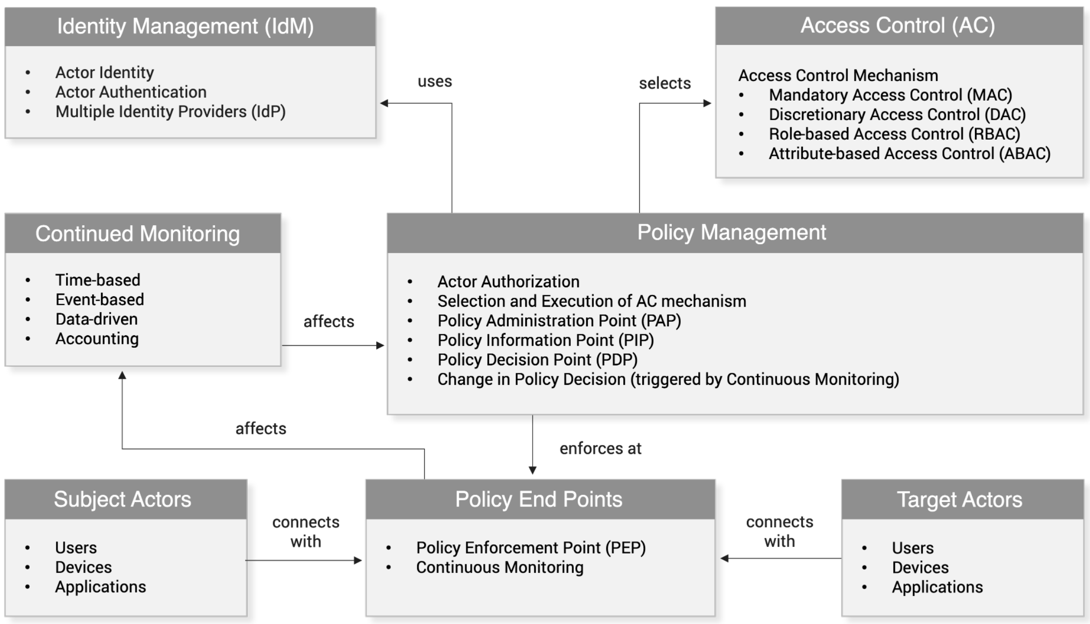
There are several standards from recognized organizations (`NIST 800-207 [11]`, `DOD Zero Trust [12]`, `CISA [13]`, `The Open Group [14]`) that can help us align Zero Trust with our typical use cases.

Basically, we're moving from a traditional perimeter-based approach to an authenticate-and-authorize everything approach.

Figure 4 Perimeter based approach to security.

Figure 3 Zero Trust approach to security.

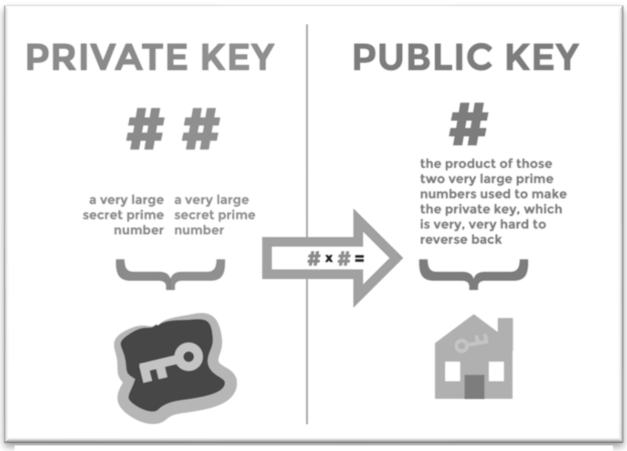
NIST says that Zero Trust is the term for an evolving set of cybersecurity paradigms that move defenses from static, network-based perimeters, to an approach focused on users, assets, and resources. Their model looks like this.


Identity Management (IdM)
Actor Identity
Actor Authentication
Multiple Identity Providers (IdP)
uses
selects
Access Control (AC)
Access Control Mechanism
Mandatory Access Control (MAC)
Discretionary Access Control (DAC)
Role-based Access Control (RBAC)
Attribute-based Access Control (ABAC)
Continued Monitoring
Time-based
Event-based
Data-driven
Accounting
affects
Policy Management
Actor Authorization
Selection and Execution of AC mechanism
Policy Administration Point (PAP)
Policy Information Point (PIP)
Policy Decision Point (PDP)
Change in Policy Decision (triggered by Continuous Monitoring)
affects
enforces at
Subject Actors
Users
Devices
Applications
connects with
Policy End Points
Policy Enforcement Point (PEP)
Continuous Monitoring
connects with
Target Actors
Users
Devices
Applications

3.2. Public and Private key pairs

Public-private key pairs are a fundamental component of asymmetric cryptography, a type of cryptographic system that uses two distinct keys: a *public key* and a *private key*. In one common use case, the public key is used to encrypt data, ensuring that only the holder of the corresponding private key can decrypt and access the information, thus securing data transmission to a specific recipient. Another important use case is digital signatures, where the private key is used to sign data. This signature can be verified by anyone with the corresponding public key, guaranteeing the authenticity and integrity of the data by confirming it came from the holder of the private key and has not been altered. This dual functionality ensures both secure communication and authentication in digital interactions.

The method of calculating public-private key pairs relies on complex mathematical problems that are easy to perform in one direction but extremely difficult to reverse. This is primarily based on the properties of certain mathematical functions, such as the factoring of large prime numbers or the discrete logarithm problem. For instance, in RSA encryption, the keys are generated from two large prime numbers multiplied together. While it is straightforward to multiply these primes to produce the public key, factoring the resultant large number to retrieve the original primes (and thus the private key) is computationally infeasible with current technology. This one-way function ensures that, even with the public key widely available, deducing the private key remains practically impossible due to the enormous computational power and time required, thus maintaining the security and integrity of the cryptographic system.


PRIVATE KEY
# #
a very large secret prime number
a very large secret prime number
# × # =
PUBLIC KEY
#
the product of those two very large prime numbers used to make the private key, which is very, very hard to reverse back
Figure 5 Generating Key Pairs

Every digital certificate contains a public key. The private key is kept secret by the certificate holder and in our case is stored in the PKCS#12 file (keychain file) owned by the certificate holder.

3.3. How Certificates guarantee authenticity

A digital certificate, like a real-world certificate, is simply a document that contains some claims, that is signed by an authority that the person relying on the certificate, trusts.

The authority in the digital realm is called the *Certificate Authority* (CA), and it can be part of a chain of authority leading back to the *Root CA*. So, a Root CA can certify that CA-1 is authentic, then CA-1 can certify that CA-2 is authentic, etc. And finally, a client or server can use a certificate signed by CA-2 because it trusts the CA-2 through the chain of trust linking back to the Root CA.

X.509 certificates use a public / private key pair. A certificate that states that the user is called Alice (a *claim*), that is subsequently encoded (*signed*) using the private key of the CA, can only be decoded by the public key of the CA. This means that we can be sure that the certificate was signed by the CA because its public key decodes it. As the public key is visible to all, anyone can verify the certificate.

In this way digital certificates allow us to assert claims that can be verified by anyone who trusts the CA that signs the claim.

Clearly in this case we need to protect the CA’s signing service so that malicious users can’t sign any fake information to provide fake certificates. So, we will need to provide services that will help EPICS operators create certificates and manage their safe storage and distribution.

Another important note is that anyone with the certificate can pretend to be the identity that is claimed in the certificate (*masquerading*). So, we need to make sure that these certificates are kept safe and secure.

3.4. Digital Certificates

There are a few certificate standards in use in computer systems but, by far, the most widely used is the X.509 (`RFC 5280 [15]`) standard. There are two other main certificate formats that could be considered – SPKI (`RFC 2692 [16] & RFC 2693 [17]`) and OpenPGP (`RFC 4880 [18]`). All the other certificates can be represented as *`X.509`* certificates which are generic certificates.

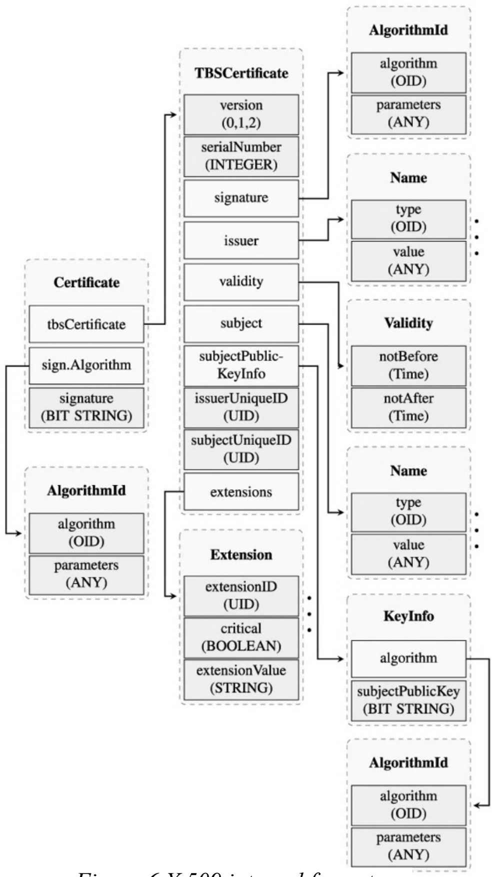


Figure 6 X.509 internal format.

The X.509 standard is backwards compatible in that clients that support prior versions, can, and will ignore any parts of the format that they don't understand. Version 3 is the latest version and contains the optional extensions.

We will support only X.509 version 3 certificates.

There are two main KeyStore formats; PKCS#12 (`RFC 7292 [9]`) for all programming languages; and JKS for Java. There are lesser-known ones, BKS (Android), Windows-My/Windows-ROOT (Windows), KeychainStore (Mac). For maximum interoperability we will use PKCS#12 and the JKS KeyStore.

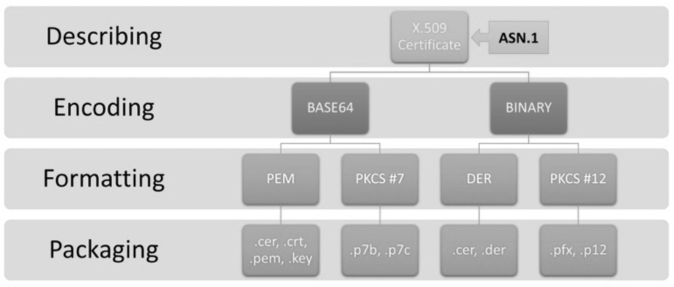


Figure 7 X.509 encoding and packaging formats.

You may have seen *`X.509`* Certificates contained in various formats: `PEM` files (Certificate files (`.CER`, `CRT`), Key files (`.KEY`), PKCS#12 files (`.P12`), and also in KeyStores (`.JKS`). They all adhere to the *`X.509`* standard in either binary or BASE64 encoding. The most appropriate for TLS are the *`PKCS#12`* and KeyStore formats which allows grouping of many components (public and private keys) with the certificate claims in one container. We will use PEM files to install the root certificate in EPICS agents so that they can trust them.

3.4.1. X.509 Server Certificates

The Key Usage part of the server certificate needs to be set to *`digitalSignature,`* and *`keyEncipherment`* while the Extended Usage needs to be set to *`serverAuth`*. These settings allow it to be used to sign any text to be sent to the client, to allow it to encode the PVAccess message payload using the session key, to send the session secret to the client to secure the communications channel, and to be used for Server Authentication.

3.4.2. X.509 Client Certificate

The Client certificate is the same format as the Server certificate except that the key usages encoded in the Key Usage section will need to be set to `digitalSignature`, and `keyEncipherment` while the Extended Usage needs to be set to `clientAuth`. These settings allow it to be used to sign any text to be sent to the server, to allow it to encode the PVAccess message payload using the session key, to send the session pre-secret to the server during TLS handshake, to be used for Client Authentication and for distinguishing the client for a client-only role.

3.4.3. Secure PVAccess custom X.509 Certificate Extension

We employ a custom certificate for our X.509 certificates that give the PV name to contact to get that certificate's status. Each PV will have a unique PV to use to get status. This custom extension is optional but if present will mean that EPICS agents will monitor status continually and adjust their connection based on changes to the certificate status. This means that if a PV is being monitored and the certificate is revoked the connection will revert to an unsecured connection immediately, and if the server serving the PV doesn't allow unsecured connections the connection will be terminated. The extension's ID is `1.3.6.1.4.1.37427.1`, its short name is `ASN.1 – PvaCertStatusURI`, and its long name is `EPICS PVA Certificate Status URI`.

3.5. OCSP

OCSP stands for Online Certificate Status Protocol, which is an internet protocol used for obtaining the revocation status of an X.509 digital certificate as well as other certificate status information. It is a more efficient method than the traditional Certificate Revocation List (CRL) method.

3.6. Network Protocol Encapsulation

Network protocols use encapsulation to add layers performing different tasks to their protocol stack. For example, the IP protocol is encapsulated inside the TCP protocol to provide TCP/IP communications.

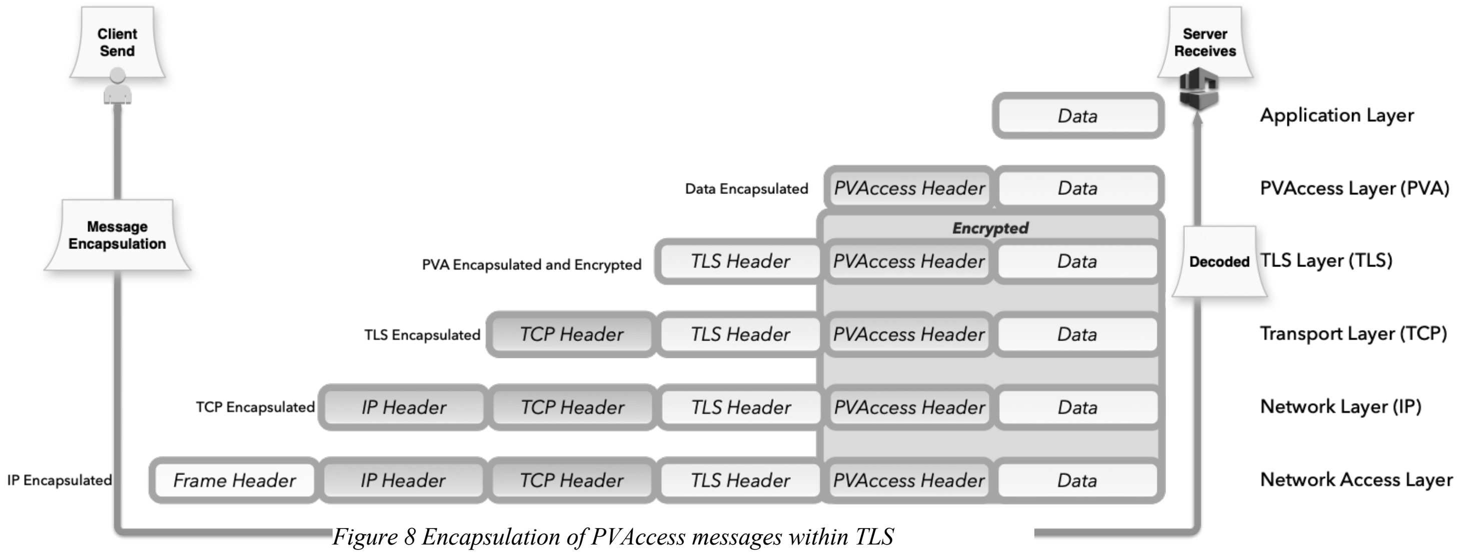


Figure 8 Encapsulation of PVAccess messages within TLS

As you can see from this diagram, when the framework gets data from the client it encapsulates it in a PVAccess message, subsequently the PVAccess message is encapsulated within TLS, and so on down the protocol stack to the physical network, the server side reverses the encapsulation process to decode the message in the TLS layer and pass on the clear PVAccess message to the Secure PVAccess server.

4. Problem Statement

The primary focus of this work is to *enable the use of TLS* in an EPICS PVAccess Protocol exchange, where the server and client supports it, and otherwise fall back to the nominal unsecured exchange.

The products of this work must *securely authenticate servers, and optionally clients*, in an EPICS network, and *securely encrypt communications* between them.

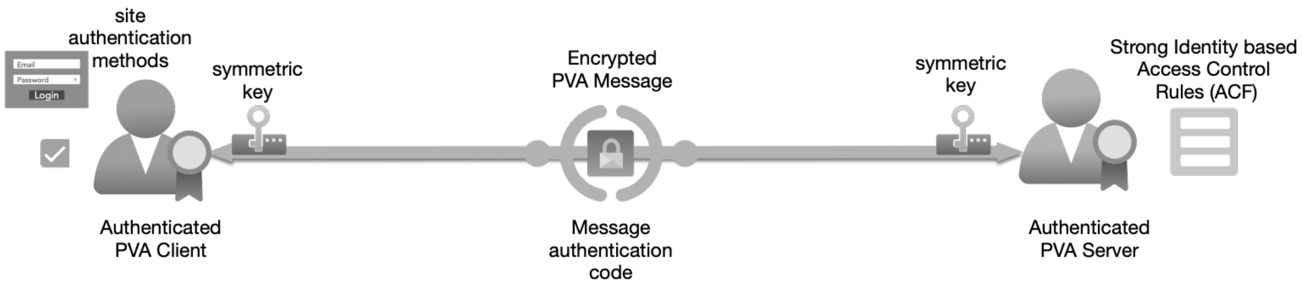


Figure 9 Secure PVAccess Session

It is a base assumption that the implementation will be done in the `PVXS code line [19]` for C++ clients and servers, and in `Phoebus [20]` for java. Modification of other EPICS client and server implementations are not in scope.

The PVAccess Protocol additions to enable this are taken to be those listed in `Secure PVAccess Protocol Proposal [3]` with additional protocol optimizations specified in this document.

A *certificate management service* must be developed (PVACMS). This should be tested with selected EPICS installations in the U.S. to evaluate usability, performance, scalability, and functionality.

The *system must integrate the authenticated clients with EPICS Security authorizations* as that mechanism exists (that is, user authorization for actions carried out on a Process Variable (PV)). The identities must be propagated securely in communications with zero trust – i.e. the server will do its own verification of the security credentials presented by the client in the certificate. This must include using X.509 client certificates, and credentials obtained from *integration with supported authentication methods*.

A solution to *automatically provision certificates* must be proposed that will be, both efficient and secure, and allow for revocation of the certificates as well as providing certificate status. A *notification service must be provided* that will allow interaction between network administrators and the certificate management service.

PVANOTIFY

5. Architecture Summary

5.1. Architecture Overview

This diagram shows the components of the proposed system.

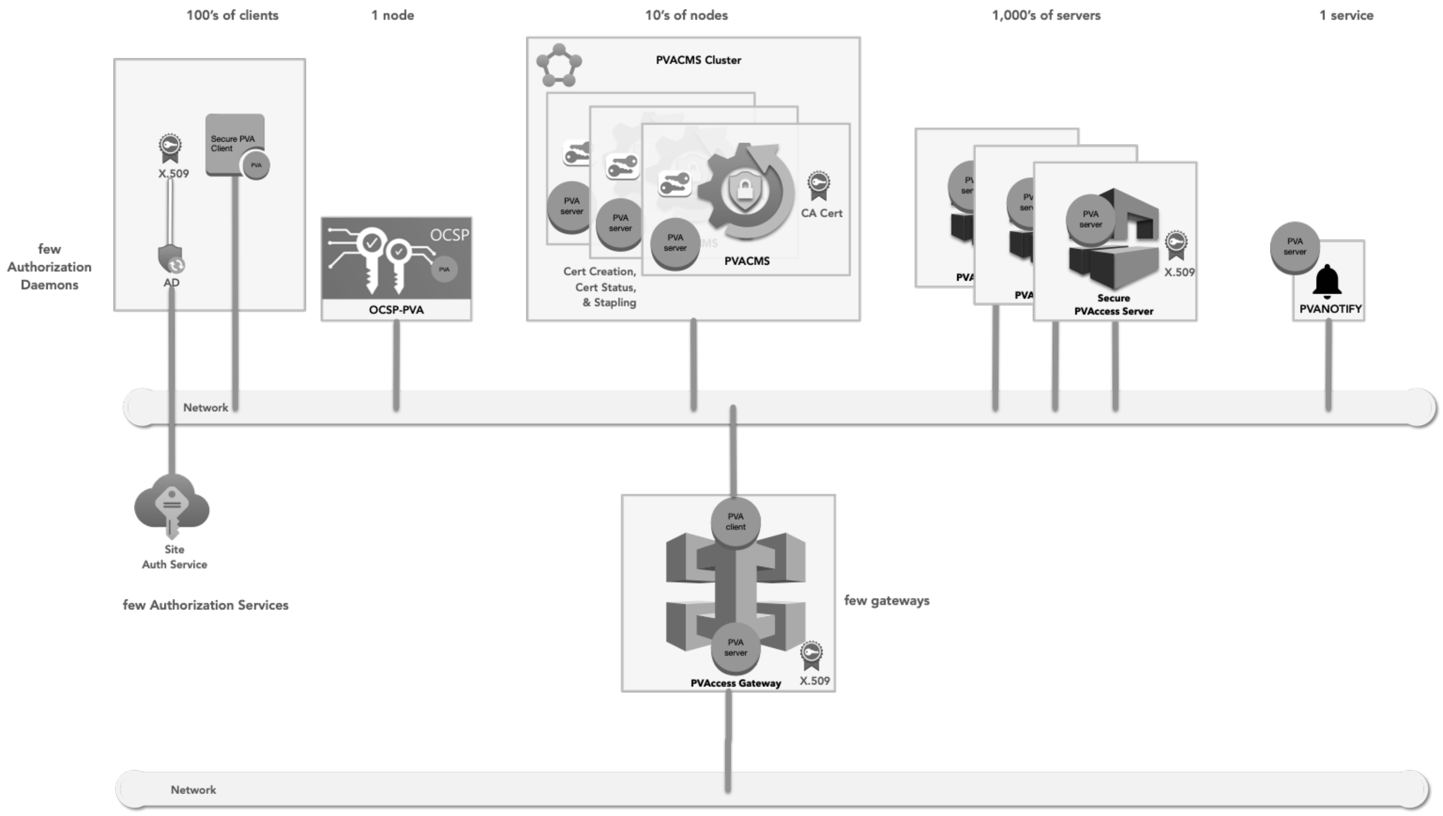


Figure 10 System Components

This diagram shows the way system components work together to provide access to control system resources in a backwards compatible way.

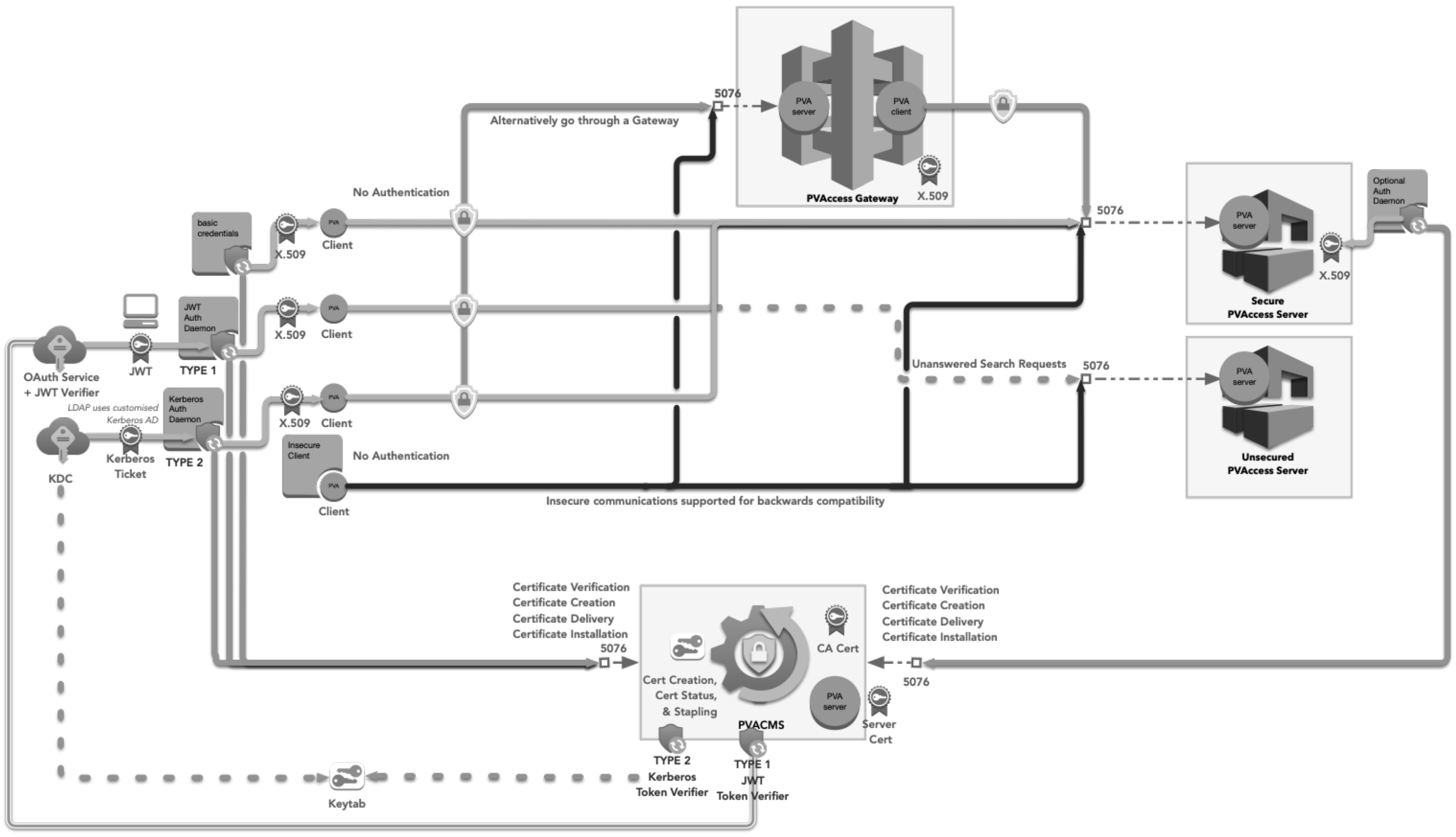


Figure 11 Network resource access in proposed system

5.2. High Level Design Topics

5.2.1. Backwards Compatibility

Any client or server that has not upgraded to the new Secure PVAccess will be able to interoperate with a client or server that has been updated. It must be able to interoperate with no degradation of service and with the same set of functions available. This must be accomplished without requiring the legacy clients or servers to recompile or change in any way, nor must they be required to change their configuration in any way.

This has been, and will remain, an important design criterion for the changes we are making. Any changes to the protocol need to be neutral to legacy systems. Any configuration required for TLS must be invisible to legacy systems. No ports must change for legacy systems. Messages that come from Secure PVAccess clients on the same sockets that the legacy systems are listening on must not cause them to fail. Incomprehensible messages must be ignored. The system must not enter a failure state when incomprehensible messages are received. Replies to Search Messages must be understandable by both legacy and new clients, and the initial Search Message must not need to change for legacy clients. Legacy Search Messages must be understood and responded to by Secure PVAccess servers.

To summarize, *PVAccess and CA Clients must be oblivious to the presence of Secure PVAccess Clients and Servers* deployed on their networks.

5.2.1.1. Beacons

A Beacon is a UDP broadcast sent by an EPICS Server. That is, PVAccess and CA servers send out beacon messages. They contain the IP Address and TCP port on which the server listens.

Historically, clients used the Beacon Messages to trigger resend of previously unanswered Search Messages. However, this practice is discouraged as there are other ways to determine the server status.

For this reason, *we will not update the Beacon messages or the Beacon functionality*. Servers will broadcast any port they want to on the Beacon messages. It is assumed that clients will not be using the ports directly but will just use the messages as an indication of server availability.

5.2.2. Authentication Methodology

Rather than designing and implementing customized TLS handshakes and verification algorithms in non-standard ways, we've opted to *always use X.509 certificates for connection* and to *exchange credentials for X.509 certificates in the Authentication Daemons*.

The main benefits of this pattern are as follows:

- *All communications will be secured by a server X.509 certificate*. All communication will conform to standards, with no hybrid TLS connections using custom encryption methods and handshake algorithms. The risk of the design hiding security flaws will be minimized. The provided mutual or server-only authentication will adhere to standards that align with our zero-trust goals.
- *Standard SSL implementation*. The solution can benefit from all future updates to TLS, because it will be implemented in a standard way. Being standard also means that it can benefit from OCSP optimization (implemented as PVACS stapling) as well as other advanced TLS features.
- *Implementation for each Authentication Method will follow the same well-defined steps*. Having all authentication methods follow the same steps to establish a connection, means that initial and future authentication methods will integrate easily, and testing will be facilitated.

5.2.3. Support for External CAs

One pattern that we have considered is that a site (*SLAC*) could decide that its root certificate (***CN****: slac.stanford.edu,* ***O****: Stanford National Laboratory,* ***OU****: SLAC Certificate Authority,* ***C****: US*), to be used to sign all its internal certificates, could, itself, be signed by an external certificate authority (*DigiCert*).

The main benefit of this approach is that if PVACMS were to be configured to use such a certificate as its root CA certificate, then all certificates generated by that PVACMS would automatically be trusted by a wide range of end user devices because trust of the most popular public certificate authorities comes bundled in the OS. Without this, the root certificate would need to be distributed to all clients and those clients would need to trust that root certificate manually.

TRUST OF POPULAR ROOT CERTIFICATES IN OPERATING SYSTEMS

Windows: *Microsoft maintains its own list of trusted root certificates for all editions of Windows. The Microsoft Trusted Root Program issues, manages, and distributes these certificates. When updates to the list are available, they are retrieved automatically via Windows Update.*

macOS: *Trusted root certificates are preinstalled with macOS. Apple provides updates to the root certificate trust settings according to the Root Certificate Program through regular software updates.*

Linux: *Trust stores in Linux vary by distribution.*

- *On Ubuntu (and other Debian-based distributions), the ca-certificates package, maintained by the Debian development community, holds the bundle of certificates.*
- *Fedora (and other Red Hat based distributions) use the ca-certificates package as well, maintained by the Fedora Project.*
- *Arch Linux uses the ca-certificates-utils package.*

In all these cases, updates to the root store come through the standard software/package update mechanism of the respective distribution.

5.2.4. Operational impacts

By providing a PVACMS that can automatically provision and install certificates on all EPICS agents with the mediation of AD's running on those agents, we seek to minimize the impact of managing the significant number of certificates required to operate a secure controls network.

If a site only has the possibility of using *basic credentials,* then there will still be significant work approving the certificates generated by this method in the `PENDING_APPROVAL` state.

Depending on how the authorization rules are structured there could also be significant work authorizing principals to give them write access to control system resources. At a minimum there would be a mapping from the enterprise user directory into EPICS Security UAGs.

A one-off integration with the network's workflow management platform from the proposed `PVANOTIFY` component should be relatively straightforward. The integration should consider batch operations and the design of the approvals process.

Tool compatibility: Some network operators use *Wireshark* or other network tools to spy on PVAccess packets on the wire. We have included an environment variable to control whether logging of the TLS keys to a file takes place during the TLS handshake process so that administrators can point Wireshark to it to have it decode PVAccess messages on the wire.

Network configuration: The configuration of the Secure PVAccess network needs to be simple and follow the same style of configuration that the legacy protocol follows. Use environment variables, minimize the number of variables to set. Use default values liberally to reduce configuration burden. Allow configuration to be scripted.

5.3. Summary of Proposed Changes

- Full *backwards compatibility* and interoperability will be provided. Secure PVAccess agents and legacy PVAccess agents will coexist and cooperate on the control systems network. Legacy PVAccess clients will make unsecured connections to Secure PVAccess servers, Secure PVAccess Gateways, or legacy PVAccess servers and gateways. Legacy PVAccess servers will be able to service requests from Secure PVAccess clients though the connection will be unencrypted.
- We will *encapsulate PVAccess Protocol messages within TLS sessions* by modifying the PVAccess `SEARCH` message to allow the request of a `tls` connection, and the `SEARCH RESPONSE` message to allow acceptance of a `tls` connection.
- EPICS agents will be able to *prove their identity by providing X.509 certificates* to the TLS handshake process.
- The certificates will be used to *establish the secure encrypted TLS session* that will encapsulate the PVAccess Protocol messages.
- Long lived connections will be accommodated by certificates with long expiration periods which will include an *EPICS custom extension,* `NID_PvaCertStatusURI`, which will require that the status be constantly verified with PVACMS enabling access to be revoked, if needed, before certificate expiry.
- EPICS agents will be able to *automatically manage X.509 certificates* X.509 by running *Authentication Daemons* (ADs) which connect to the agent's authentication services. They monitor the state of the authentication and create/update/replace the certificates as needed. Provisioning of certificates will be provided by `PVACMS` that will leverage the PVAccess protocol to respond to certificate creation requests (CCR) from the ADs.
- Examples of these Authentication Daemons will be provided for *variety of authentication methods*. **Kerberos**, **LDAP** backed Kerberos, **JWT**, and **basic credentials**.
- The framework for implementing *support for authentication methods will be designed to be modular and extensible*, allowing for future enhancements and additional support.
- PVACMS will also *process certificate revocations*, *certificate approval request acceptances* and *approval request rejections* that will be available only to administrators configured in the PVACMS's ACF file.
- EPICS agents will automatically reconfigure their connections based on the availability of certificates and on certificate statuses, changing from unsecured to secure and back again as necessary.
- PVACMS will *respond to PVAccess Certificate Status requests* (PVACS requests). It will maintain a permanent certificate status that will persist beyond the process lifetime and will be replicated across the cluster in a secure and efficient manner. The status will be tied to the Subject Key Identifier (SKID) such that no duplicate certificates (with the same subject name or public key) will be allowed to be generated.
- *PVAccess Certificate Status Stapling* (PVACS stapling) will be implemented by retrieving a Secure PVAccess server's certificate status using a PVACS message and stapling the OCSP encoded PKCS#7 response to the TLS handshake so the Secure PVAccess client won't have to check the server certificate's status. The OCSP response is signed by the CA that is trusted by all clients who can therefore authenticate the status locally.
- The PVACMS executable will work in standalone mode but will automatically switch to cluster mode if another node is started on the same PVAccess network (same ADDR_LIST). The cluster will be self-healing when nodes die unexpectedly and will resize automatically when new nodes are added or removed. The cluster will provide *automatic load balancing and fault tolerance* to support hundreds of thousands of certificate creation, revocation, and status requests per second. We will not implement auto-scaling of the PVACMS cluster.
- To notify administrators and logging systems of Secure PVAccess events, and to capture responses to those events that require action, we will provide the *PVANOTIFY* service (`pvanotify`). This service will be notified whenever certificates are created, and whenever their status changes. Sites will be able to integrate with their preferred service request platform/software by adding code to the `pvanotify` sources. An example of a simple integration to Slack will be provided.
- For controls network clients that don't communicate using PVAccess but wish to leverage the certificates produced by the PVACMS we will *provide the OCSP-PVA* (`ocsp-pva`) that will offer a regular OCSP service by forwarding requests to the PVACS PVs served by the PVACMS cluster and translating results into OCSP responses.

6. Detailed Functional Specifications

6.1. EPICS Agent Environment Variables

The environment variables in the following table configure the EPICS agents at runtime.

Note: In all environment variables where there are `PVA`, and `PVAS` versions they refer to the EPICS client, and EPICS server versions of the variable respectively. e.g. if `EPICS_PVA_TLS_OPTIONS` and `EPICS_PVAS_TLS_OPTIONS` are both configured on a server where both an EPICS client and server are running then the client will get its TLS options from `EPICS_PVA_TLS_OPTIONS` and the server will get its configuration from `EPICS_PVAS_TLS_OPTIONS`.

Note: There is also an implied hierarchy to their applicability such that the `PVAS` supersedes the `PVA` version. So, if an EPICS server wants to specify its PKCS#12 keychain file location it can simply provide the `EPICS_PVA_TLS_KEYCHAIN` environment variable as long the `EPICS_PVAS_TLS_KEYCHAIN` is not configured.

NAME	Keys &	Values	Description
`EPICS_PVA_TLS_KEYCHAIN` ***`EPICS_PVAS_TLS_KEYCHAIN`***	{fully qualified path to keychain file} e.g. `~/.epics/client.p12` e.g. `~/.epics/server.p12`		This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the certificate, and public and private keys used in the TLS handshake with peers. **Note**: If not specified then TLS is disabled.

`EPICS_PVA_TLS_KEYCHAIN_ PWD_FILE`
`EPICS_PVAS_TLS_KEYCHAIN_ PWD_FILE`
	{fully qualified path to keychain password file}
	e.g. `~/.epics/client.pass`
	e.g. `~/.epics/server.pass`
			This is the string that determines the fully qualified path to a file that contains the password that unlocks the TLS KEYCHAIN file. This is optional. If not specified, the TLS KEYCHAIN file contents will not be encrypted.

`EPICS_PVA_TLS_KEY` ***`EPICS_PVAS_TLS_KEY`***	{fully qualified path to key file} e.g. `~/.epics/clientkey.p12` e.g. `~/.epics/serverkey.p12`		This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the private key used in the TLS handshake with peers. **Note**: This is optional and if not specified the TLS_KEYCHAIN file is used.

`EPICS_PVA_TLS_KEY_ PWD_FILE`
`EPICS_PVAS_TLS_KEY_ PWD_FILE`
	{fully qualified path to key password file}
	e.g. `~/.epics/clientkey.pass`
	e.g. `~/.epics/serverkey.pass`
			This is the string that determines the fully qualified path to a file that contains the password that unlocks the TLS KEY file. This is optional. If not specified, the TLS KEY file contents will not be encrypted.

`EPICS_PVA_TLS_OPTIONS` ***`EPICS_PVAS_TLS_OPTIONS`*** Set TLS options for clients and servers. A string containing key/value pairs separated by commas, tabs or newlines	`client_cert` *Determines whether client certificates are required*	`require`	Require client certificate to be presented.
		`optional`	Don't require client certificate to be presented.
	`on_ expiration` *Determines what to do when an EPICS agent's certificate has expired, and a new one can't be automatically provisioned*	`fallback-to-tcp`	For servers only tcp search requests will be responded to and if it's a client then no client certificate will be presented in the tls handshake (but searches will still offer both tls and tcp as supported protocols) .
		`shutdown`	The process will exit gracefully.
		`standby`	Servers will not respond to any requests until a new certificate is successfully provisioned. It will keep retrying auto-provisioning requests (if configured) and checking the keychain file periodically. When a valid certificate is available it will continue as normal. For a client `standby` has the same effect as `shutdown`.
`EPICS_PVA_TLS_PORT` ***`EPICS_PVAS_TLS_PORT`***	{port number} default `5076` e.g. `8076`		This is a number that determines the port used for the Secure PVAccess, either as the port on the Secure PVAccess server for clients to connect to – *PVA*, or as the local port number for Secure PVAccess servers to listen on – *PVAS.* The default is the same as the existing *`EPICS_PVAS_SERVER_PORT`* and *`EPICS_PVA_SERVER_PORT`* settings.

NAME	Keys & Values	Description

EPICS_PVAS_TLS_ STOP_IF_NO_CERT	case insensitive: `YES`, `TRUE`, or `1`
	case insensitive: `NO`, `FALSE`, or `0`
		If true, the server will stop if no valid cert is available, and no valid cert can be commissioned through all the supported methods. Use this to force a server to be able to accept TLS connections.

SSLKEYLOGFILE	{fully qualified path to key log file} e.g. `~/.epics/keylog`	This is the path to the SSL key log file that, in conjunction with the build-time macro *`PVXS_ENABLE_SSLKEYLOGFILE`*, controls where and whether we store the session key for TLS sessions in a file. If it is defined, then the code will contain the calls to save the keys in the file specified by this variable.

6.2. PVACMS *Environment Variables*

The environment variables in the following table configure the PVACMS at runtime.

Note: There is also an implied hierarchy to their applicability such that `PVACMS` supersedes the `PVAS` version which in turn, supersedes the `PVA` version. So, if a PVACMS wants to specify its PKCS#12 keychain file location it can simply provide the `EPICS_PVA_TLS_KEYCHAIN` environment variable as long as neither `EPICS_PVACMS_TLS_KEYCHAIN` nor `EPICS_PVAS_TLS_KEYCHAIN` are configured.

NAME	Keys & Values	Description
EPICS_AUTH_JWT_ REQUEST_FORMAT	{string format for verification request payload} e.g. `{ "token": "#token#" }` e.g. `#token#`	**PVACMS JWT AD only:** A string that is used verbatim as the payload for the verification request while substituting the string `#token#` for the token value, and `#kid#` for the key id. This is used when the verification server requires a formatted payload for the verification request. If the string is `#token#` (default) then the verification endpoint is called with the raw token as the payload

EPICS_AUTH_JWT_ REQUEST_METHOD	`POST` (default)
	`GET`
	
	e.g. of call made for `GET`:
	
	`GET /api/validate-token HTTP/1.1`
	`Authorization: Bearer eyJhbGcXVCJ9...`
		PVACMS JWT AD only: This determines whether the endpoint will be called with `HTTP GET` or `POST`.
		
		If called with **POST,** then the payload is exactly what is defined by the *`EPICS_AUTH_JWT_ RESPONSE_FORMAT`* variable.
		
		If called with **GET,** then the token is passed in the Authorization header of the HTTP GET request.

EPICS_AUTH_JWT_ RESPONSE_FORMAT	{string format for verification response value} e.g. `{ "payload": { * }, "valid": #response# }` e.g. `#response#`	**PVACMS JWT AD only:** A pattern string that we can use to decode the response from a verification endpoint if the response is formatted text. All white space is removed in the given string and in the response. Then all the text prior to `#response#` is matched and removed from the response and all the text after the response is likewise removed, what remains is the response value. An asterisk in the string matches any sequence of characters in the response. It is converted to lowercase and interpreted as valid if it equals `valid`, `ok`, `true`, `t`, `yes`, `y`, or `1`. If the string is `#response#` (default) then the response is raw and is converted to lowercase and compared without removing any formatting
EPICS_AUTH_JWT_TRUSTED_URI	{uri of JWT validation endpoint} e.g. `http://ssuer/api/validate-token`	**PVACMS JWT AD only:** Trusted URI of the validation endpoint – the substring that starts the URI including the `http://`, `https://` and port number. There is no default, it must be specified. This is used to compare to the *`iss`* field in the decoded token payload if it is provided. If it is not the same, then the validation fails. If the *`iss`* field

NAME	Keys & Values	Description
		is missing, then the value of this variable is taken as the validation URI.

EPICS_AUTH_JWT_ USE_RESPONSE_CODE	case insensitive: `YES`, `TRUE`, or `1`
	case insensitive: `NO`, `FALSE`, or `0`
		PVACMS JWT AD only: If set this tells PVACMS that when it receives a `200` HTTP-response code from the HTTP request then the token is valid, and invalid for any other response code.

EPICS_AUTH_KRB_KEYTAB	{fully qualified path to PVACMS/CLUSTER Kerberos service keytab file} e.g. `/etc/krb5/keytab` e.g. `~/.epics/pvacms.keytab`	**PVACMS KRB AD only**: This string is the fully qualified path to the location of the `keytab` file. It is used to retrieve the secret key used to decode messages destined for a Kerberos service
EPICS_AUTH_LDAP_ACCOUNT	{distinguished name for an account with sufficient permissions to query the LDAP directory} e.g. `"cn=admin,dc=slac,dc=stanford,dc=edu"`	**PVACMS LDAP AD only**: distinguished name for an account with sufficient permissions to query the LDAP directory.
EPICS_AUTH_LDAP_ACCOUNT_ PWD_FILE	{fully qualified path to LDAP account password file} e.g. `~/.epics/ldap.pass`	**PVACMS LDAP AD only**: This is the string that determines the fully qualified path to a file that contains the password for the configured LDAP account.
EPICS_AUTH_LDAP_HOST	{hostname or IP address of the LDAP server} e.g. `ldap://ldap.slac.stanford.edu`	**PVACMS LDAP AD only**: hostname or IP address of the LDAP server. This server will be queried to determine if the principal obtained by GSS-API comes from the directory store. This must be specified for the LDAP authentication method.
EPICS_AUTH_LDAP_PORT	{port number of configured LDAP host} e.g. `389` (default `389`)	**PVACMS LDAP AD only**: this is the port number to contact the LDAP service on.
EPICS_AUTH_LDAP_SEARCH_ROOT	{distinguished name for root of LDAP directory search} e.g. `"cn=slac,dc=stanford,dc=edu"`	**PVACMS LDAP AD only**: distinguished name for location within LDAP directory to start search.

EXAMPLE ACF for REVOKE AUTHORIZATION
`UAG(admins) {`
`    "george",`
`    "michael",`
`    "role/admin"`
`}`
`ASG(RESTRICT) {`
`    RULE(1, WRITE, TRAPWRITE) {`
`        UAG(special)`
`        METHOD(x509)`
`    }`
`}`
EPICS_CA_ACF
	{fully qualified path to PVACMS ACF file}
	e.g. `~/.epics/pvacms.acf`
		This is the string that determines the fully qualified path to a file that will be used as the ACF file that configures the permissions that are accorded to validated peers of the PVACMS. This will specify administrators that have the right to revoke certificates, and the default read permissions for certificate statuses. There is no default so it must be specified on the command line or as an environment variable.

EPICS_CA_DB	{fully qualified path to *sqlite* certificate database file} e.g. `~/.epics/certs.db`	This is the string that determines the fully qualified path to a file that will be used as the *sqlite* PVACMS certificate database for a PVACMS process. The default is the current directory in a file called `certs.db`
EPICS_CA_KEYCHAIN	{fully qualified path to keychain file} e.g. `~/.epics/ca.p12`	This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the CA certificate, and public and private keys. This is used to sign certificates being created in the PVACMS and sign certificate status payloads. **Note**: This certificate needs to be trusted by all EPICS agents. This can be set as a command line parameter. e.g. `-c ~/.epics/ca.p12`

NAME	Keys & Values	Description
EPICS_CA_KEYCHAIN_PWD_FILE	{fully qualified path to *EPICS_CA_KEYCHAIN* password file} e.g. `~/.epics/ca.pass`	**PVACMS and OCSP-PVA only**: This is the string that determines the fully qualified path to a file that contains the password that unlocks the *EPICS_CA_KEYCHAIN* file. This is optional. If not specified, the *EPICS_CA_KEYCHAIN* file contents will not be encrypted.
EPICS_CA_KEY	{fully qualified path to key file} e.g. `~/.epics/cakey.p12`	This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the private key. This is used to sign certificates being created in the PVACMS and sign certificate status payloads. **Note**: This is optional and may be stored in the same file as the KEYCHAIN.
EPICS_CA_KEY_PWD_FILE	{fully qualified path to *EPICS_CA_KEY* password file} e.g. `~/.epics/cakey.pass`	This is the string that determines the fully qualified path to a file that contains the password that unlocks the *EPICS_CA_KEY* file. This is optional.
EPICS_CA_NAME	{string}	If a CA root certificate has not been established prior to the first time that the PVACMS starts up, then one will be created automatically. To provide the name (**CN**) to be used in the *subject* of the CA certificate we can use this environment variable, or we can pass the **-n** command line argument. e.g. **-n** `"PVACMS"`.
EPICS_CA_ORGANIZATION	{string}	If a CA root certificate has not been established prior to the first time that the PVACMS starts up, then one will be created automatically. To provide the organization (**O**) to be used in the subject of the CA certificate we can use this environment variable, or we can pass the `-o` command line argument. e.g. `-o "SLAC.STANFORD.EDU"`.
EPICS_CA_ ORGANIZATIONAL_UNIT	{string}	If a CA root certificate has not been established prior to the first time that the PVACMS starts up, then one will be created automatically. To provide the organizational unit (**OU**) to be used in the subject of the CA certificate we can use this environment variable, or we can pass the `-u` command line argument. e.g. `-u "SLAC Authorization Service"`.

EPICS_CA_ STATUS_CHECK_OPTIONAL	case insensitive: `YES`, `TRUE`, or `1`
	case insensitive: `NO`, `FALSE`, or `0`
	(default **`false`**)
		To determine whether to add the custom attribute to all certificates generated that controls whether the certificates require a status check before their validity is affirmed. If set then the PV of the status for that certificate is added as a custom extension to the certificate.

EPICS_PVACMS_ REQUIRE_CLIENT_APPROVAL	case insensitive: `YES`, `TRUE`, or `1`
	case insensitive: `NO`, `FALSE`, or `0`
	(default **`false`**)
		When basic credentials are used then set to **`true`** to request administrator approval to use client certificates. This will mean that clients will receive a certificate that is in the `PENDING_AUTHORIZATION` state.

EPICS_PVACMS_ REQUIRE_SERVER_APPROVAL	case insensitive: `YES`, `TRUE`, or `1`
	case insensitive: `NO`, `FALSE`, or `0`
	(default **`true`**)
		When basic credentials are used then set to **`true`** to request administrator approval to issue server certificates. This will mean that servers will receive a certificate that is in the `PENDING_AUTHORIZATION` state.

NAME	Keys & Values	Description
EPICS_PVAS_TLS_KEYCHAIN *EPICS_PVACMS_TLS_KEYCHAIN*	{fully qualified path to keychain file} .g. `~/.epics/server.p12` e.g. `~/.epics/pvacms.p12`	This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the certificate, and public and private keys used in the TLS handshake with peers. **Note**: If not specified then TLS is disabled.
EPICS_PVAS_TLS_KEYCHAIN_ PWD_FILE *EPICS_PVACMS_TLS_KEYCHAIN_ PWD_FILE*	{fully qualified path to keychain password file} e.g. `~/.epics/server.pass` e.g. `~/.epics/pvacms.pass`	This is the string that determines the fully qualified path to a file that contains the password that unlocks the TLS KEYCHAIN file. This is optional. If not specified, the TLS KEYCHAIN file contents will not be encrypted.
EPICS_PVAS_TLS_PORT	{port number} default `5076` e.g. `8076`	This is a number that determines the port used for the Secure PVAccess, as the local port number for Secure PVAccess servers to listen on. The default is the same as the existing *EPICS_PVAS_SERVER_PORT* and *EPICS_PVACMS_SERVER_PORT* settings.
EPICS_PVACMS_TLS_PORT		
EPICS_PVAS_TLS_ STOP_IF_NO_CERT	case insensitive: `YES`, `TRUE`, or `1` case insensitive: `NO`, `FALSE`, or `0`	If true, the server will stop if no valid cert is available, and no valid cert can be commissioned through all the supported methods. Use this to force a server to be able to accept TLS connections.
EPICS_PVACMS_TLS_ STOP_IF_NO_CERT		
SSLKEYLOGFILE	{fully qualified path to key log file} e.g. `~/.epics/keylog`	This is the path to the SSL key log file that, in conjunction with the build-time macro *PVXS_ENABLE_SSLKEYLOGFILE*, controls where and whether we store the session key for TLS sessions in a file. If it is defined, then the code will contain the calls to save the keys in the file specified by this variable.

6.3. JWT AD *Environment Variables*

The environment variables in the following table configure the JWT Authentication Daemon at runtime.

NAME	Keys & Values	Description
EPICS_AUTH_JWT_ REQUEST_FORMAT	{string format for verification request payload} e.g. `{ "token": "#token#" }` e.g. `#token#`	**PVACMS only:** A string that is used verbatim as the payload for the verification request while substituting the string `#token#` for the token value, and `#kid#` for the key id. This is used when the verification server requires a formatted payload for the verification request. If the string is `#token#` (default) then the verification endpoint is called with the raw token as the payload
EPICS_AUTH_JWT_ REQUEST_METHOD	`POST` (default) `GET` e.g. of call made for `GET`: `GET /api/validate-token HTTP/1.1` `Authorization: Bearer eyJhbGcXVCJ9...`	**PVACMS only**: This determines whether the endpoint will be called with `HTTP GET` or `POST`. If called with **POST,** then the payload is exactly what is defined by the *EPICS_AUTH_JWT_ RESPONSE_FORMAT* variable. If called with **GET,** then the token is passed in the Authorization header of the HTTP GET request.
EPICS_AUTH_JWT_ RESPONSE_FORMAT	{string format for verification response value} e.g. `{ "payload": { * }, "valid": #response# }` e.g. `#response#`	**PVACMS only:** A pattern string that we can use to decode the response from a verification endpoint if the response is formatted text. All white space is removed in the given string and in the response. Then all the text prior to `#response#` is matched and removed from the response and all the text after the response is likewise removed, what remains is the response value. An asterisk in the string matches

NAME	Keys & Values	Description
		any sequence of characters in the response. It is converted to lowercase and interpreted as valid if it equals `valid`, `ok`, `true`, `t`, `yes`, `y`, or `1`. If the string is `#response#` (default) then the response is raw and is converted to lowercase and compared without removing any formatting
EPICS_AUTH_JWT_TRUSTED_URI	{uri of JWT validation endpoint} e.g. `http://ssuer/api/validate-token`	**PVACMS only:** Trusted URI of the validation endpoint – the substring that starts the URI including the `http://`, `https://` and port number. There is no default, it must be specified. This is used to compare to the `iss` field in the decoded token payload if it is provided. If it is not the same, then the validation fails. If the `iss` field is missing, then the value of this variable is taken as the validation URI.

EPICS_AUTH_JWT_ USE_RESPONSE_CODE	case insensitive: `YES`, `TRUE`, or `1`
	case insensitive: `NO`, `FALSE`, or `0`
		PVACMS only: If set this tells PVACMS that when it receives a `200` HTTP-response code from the HTTP request then the token is valid, and invalid for any other response code.

EPICS_PVA_TLS_KEY	{fully qualified path to key file} e.g. `~/.epics/clientkey.p12`	This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the private key used to generate a new certificate.
EPICS_PVAS_TLS_KEY	e.g. `~/.epics/serverkey.p12`	
EPICS_PVA_TLS_KEY_ PWD_FILE	{fully qualified path to key password file} e.g. `~/.epics/clientkey.pass`	This is the string that determines the fully qualified path to a file that contains the password that unlocks the TLS KEY file. This is optional. If not specified, the TLS KEY file contents will not be encrypted.
EPICS_PVAS_TLS_KEY_ PWD_FILE	e.g. `~/.epics/serverkey.pass`	

6.4. KRB AD *Environment Variables*

The environment variables in the following table configure the Kerberos Authentication Daemons at runtime.

NAME	Keys & Values	Description
EPICS_AUTH_KRB_REALM	{string realm of PVACMS/CLUSTER service} e.g. `SLAC.STANFORD.EDU` ➔ `PVACMS/CLUSTER@SLAC.STANFORD.EDU`	It is used in an EPICS agent when creating a **GSSAPI** context to create a token to send to the PVACMS to be validated. There is no default so this value **must** be specified if Kerberos support is configured.
EPICS_PVA_TLS_KEY	{fully qualified path to key file} e.g. `~/.epics/clientkey.p12`	This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the private key used to generate a new certificate.
EPICS_PVAS_TLS_KEY	e.g. `~/.epics/serverkey.p12`	
EPICS_PVA_TLS_KEY_ PWD_FILE	{fully qualified path to key password file} e.g. `~/.epics/clientkey.pass`	This is the string that determines the fully qualified path to a file that contains the password that unlocks the TLS KEY file. This is optional. If not specified, the TLS KEY file contents will not be encrypted.
EPICS_PVAS_TLS_KEY_ PWD_FILE	e.g. `~/.epics/serverkey.pass`	

6.5. LDAP AD *Environment Variables*

The environment variables in the following table configure the Kerberos Authentication Daemons at runtime.

NAME	Keys & Values	Description
EPICS_PVA_TLS_KEY	{fully qualified path to key file} e.g. `~/.epics/clientkey.p12`	This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the private key used to generate a new certificate.
EPICS_PVAS_TLS_KEY	e.g. `~/.epics/serverkey.p12`	
EPICS_PVA_TLS_KEY_ PWD_FILE	{fully qualified path to key password file} e.g. `~/.epics/clientkey.pass`	This is the string that determines the fully qualified path to a file that contains the password that unlocks

NAME	Keys & Values	Description
EPICS_PVAS_TLS_KEY_ PWD_FILE	e.g. ~/.epics/serverkey.pass	the TLS KEY file. This is optional. If not specified, the TLS KEY file contents will not be encrypted.

6.6. Basic Credentials AD *Environment Variables*

The environment variables in the following table configure the basic credentials Authorization Daemon at runtime.

NAME	Keys & Values	Description
EPICS_AUTH_STD_ CERT_VALIDITY_MINS	{number of minutes for cert validity} e.g. 43200 for 30 days	The number of minutes from now after which the new certificate being created should expire. Use this to set the default validity for certificates
EPICS_AUTH_STD_DEVICE_NAME	{case sensitive device name} e.g. "RTC01"	Value will be used as the device name when an AD is determining basic credentials instead of the hostname as the principal.
EPICS_AUTH_STD_PROCESS_NAME EPICS_AUTH_STD_PROCESS_NAME	{case sensitive process name} e.g. "archiver"	Value will be used as the process name when an AD is determining basic credentials instead of the logged-on user as the principal.

EPICS_AUTH_STD_ USE_PROCESS_NAME	case insensitive: YES, TRUE, or 1
	case insensitive: NO, FALSE, or 0
		true will mean that when the AD is determining the basic credentials it will use the process name instead of the logged-on user as the principal.

EPICS_PVA_TLS_KEY EPICS_PVAS_TLS_KEY	{fully qualified path to key file} e.g. ~/.epics/clientkey.p12 e.g. ~/.epics/serverkey.p12	This is the string that determines the fully qualified path to the PKCS#12 keychain file that contains the private key used to generate a new certificate.
EPICS_PVA_TLS_KEY_ PWD_FILE EPICS_PVAS_TLS_KEY_ PWD_FILE	{fully qualified path to key password file} e.g. ~/.epics/clientkey.pass e.g. ~/.epics/serverkey.pass	This is the string that determines the fully qualified path to a file that contains the password that unlocks the TLS KEY file. This is optional. If not specified, the TLS KEY file contents will not be encrypted.

6.7. OCSP *Environment Variables*

The environment variables in the following table configure the OCSPPVA process at runtime.

NAME	Keys & Values	Description
EPICS_OCSP_PORT	{port number} default 8080 e.g. 8121	**OCSP-PVA only:** The port for the OCSP server to listen on. Defaults to 8080.

6.8. Securing the PVAccess Protocol

6.8.1. *TLS Implementation*

The proposed protocol sequence to establish a Secure PVAccess connection is shown below in the simplified sequence diagram.

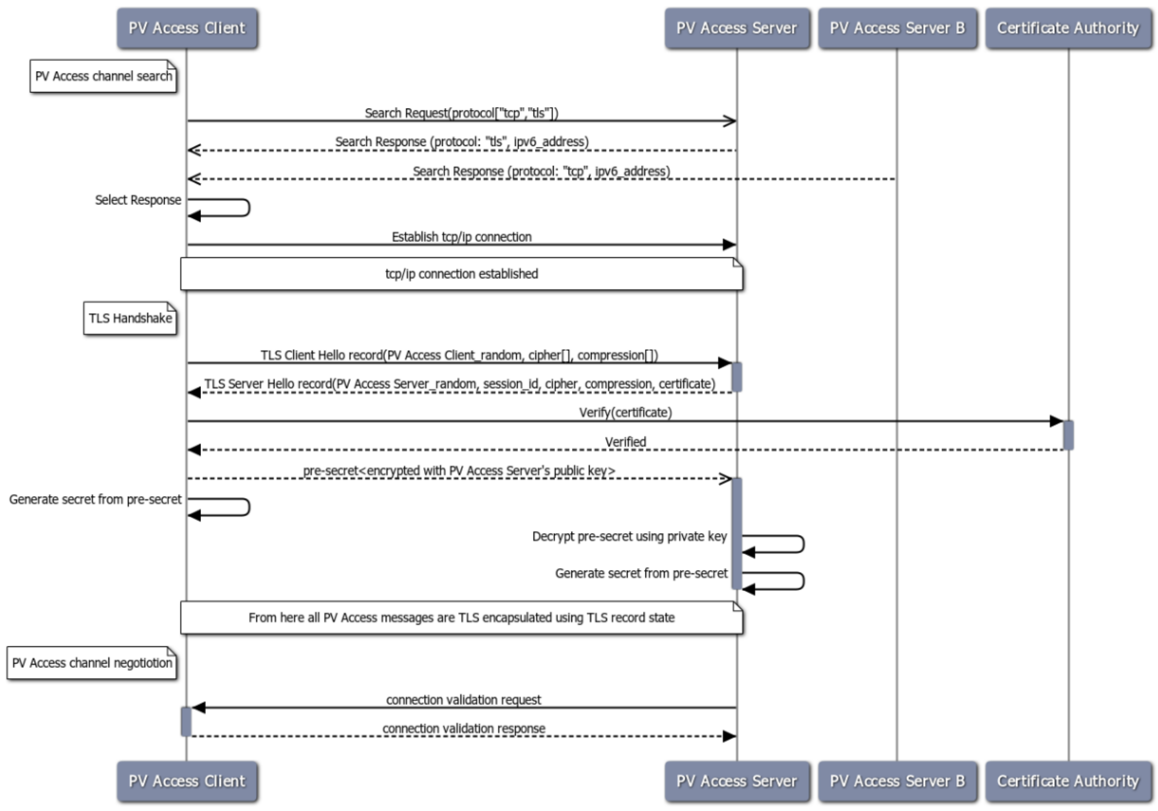
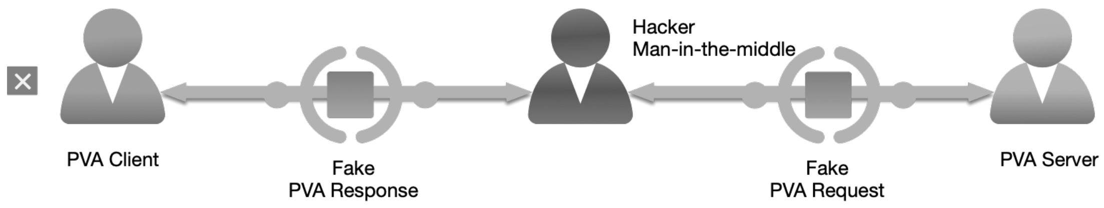

PV Access Client
PV Access Server
PV Access Server B
Certificate Authority
PV Access channel search
Search Request(protocol["tcp","tls"])
Search Response (protocol: "tls", ipv6_address)
Search Response (protocol: "tcp", ipv6_address)
Select Response
Establish tcp/ip connection
tcp/ip connection established
TLS Handshake
TLS Client Hello record(PV Access Client_random, cipher[], compression[])
TLS Server Hello record(PV Access Server_random, session_id, cipher, compression, certificate)
Verify(certificate)
Verified
pre-secret<encrypted with PV Access Server's public key>
Generate secret from pre-secret
Decrypt pre-secret using private key
Generate secret from pre-secret
From here all PV Access messages are TLS encapsulated using TLS record state
PV Access channel negotiotion
connection validation request
connection validation response
PV Access Client
PV Access Server
PV Access Server B
Certificate Authority

Figure 12 Basic TLS Protocol Sequence

6.8.2. SEARCH and SEARCH RESPONSE messages

Search – `SEARCH` messages are sent by PVAccess clients to search for a PVAccess Server that can respond to requests on a particular PV. In the search message the client will advertise the list of protocols it supports. In the current version of PVAccess this is only *`tcp`*. In the Secure PVAccess version we propose *adding `tls` as the name of a new protocol* that could be advertised as supported by the client.

Vulnerability: This means that a legacy PVAccess Server could reply before a Secure PVAccess Server, thus establishing an unsecured connection. This would allow a malicious user to initiate a man-in-the-middle attack, by masquerading as the Secure PVAccess Server. Our proposal mitigates the risk by encoding the **method**, **authority**, and unauthorized **credentials** into the protocol's `AUTHZ` message (and verifying these fields on the Secure PVAccess server before using for authorization) allowing network administrators to block such attempts.



Search Response – `SEARCH RESPONSE` messages. A server that recognizes the PV in the search message will respond with the protocol that it supports. It returns *`tcp`* if it is not configured for TLS and *`tls`* if it is.

6.8.3. URI scheme

In a fully qualified PV URI, the PV **name** is preceded by a **scheme**. This indicates to the PVAccess server what protocol scheme is being used to access the PV. Currently there is only *`pva`*, but we will add *`pvas`* as an allowed scheme to be used for all TLS (*`tls`*) connections. As the scheme is optional there won't be any backwards-compatibility issues.

6.8.4. Ports

We will use a different port for Secure PVAccess and will add a new environment variable so that it can be configured separately.

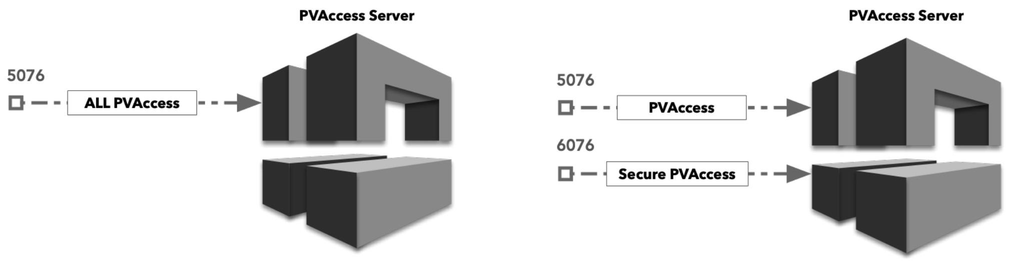


Figure 13 Port usage options for Secure PVAccess

6.9. Authentication

Authentication of EPICS agents is the main feature enhancement that this work brings to the PVAccess Protocol. It does it by requiring agents to present a certificate if they want to participate in a **mutually authenticated** secure communication with a peer over Secure PVAccess. The proposal also permits **unilaterally** (**server-only**) **authenticated** sessions.

For Kerberos we will use the AD detailed in *6.10.4.1. Kerberos* Service Ticket support below. For clients that have a JWT obtained by an OAuth exchange, we will use the AD outlined in *6.10.4.3. JWT token and OAuth Support* below. For Kerberos on LDAP see *6.10.4.2. Kerberos on LDAP* support below. For sites that don't use a supported authentication method a **basic-credentials** AD is available. See *6.10.4.4. Basic Credentials* below for how X.509 certificates can be obtained using basic-credentials.

6.9.1. Server and Client Startup

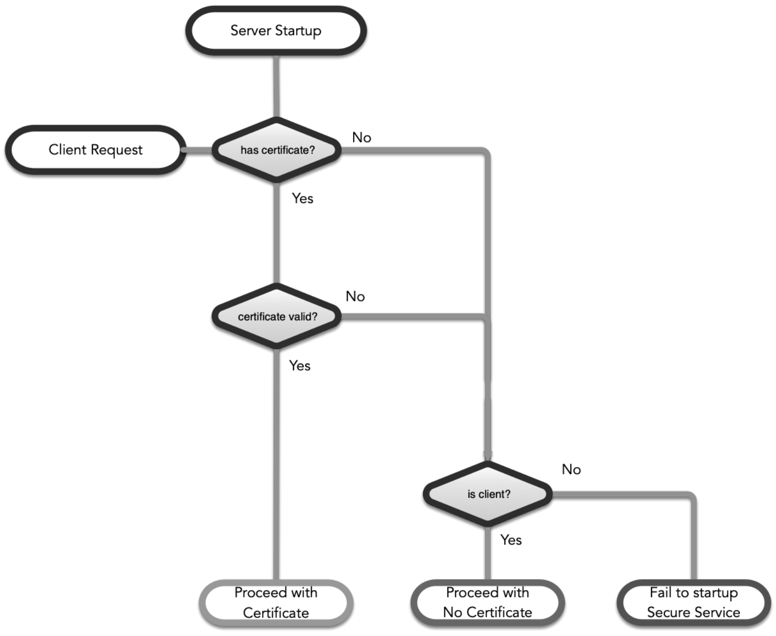


Figure 14 Server and Client certificate acquisition startup

6.9.2. Authentication Integration Framework

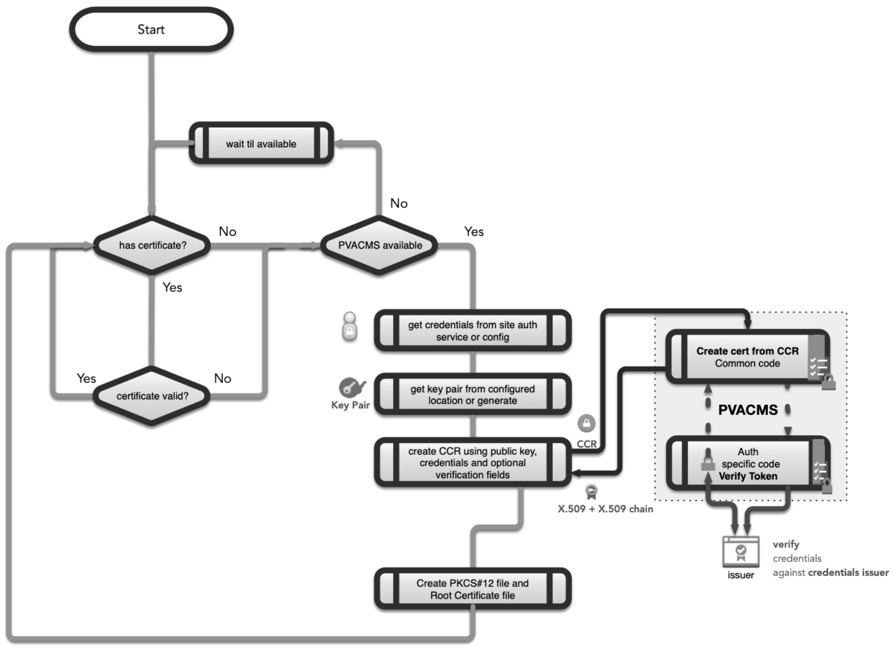

Start
wait til available
No
has certificate?
No
PVACMS available
Yes
Yes
get credentials from site auth service or config
Key Pair
get key pair from configured location or generate
Yes
certificate valid?
No
create CCR using public key, credentials and optional verification fields
CCR
X.509 + X.509 chain
Create cert from CCR Common code
PVACMS
Auth specific code Verify Token
issuer
verify credentials against credentials issuer
Create PKCS#12 file and Root Certificate file

Figure 15 Authentication Integration Framework

The proposed framework supports pluggable authentication methods that gather credentials, allow these credentials to be verified by the PVACMS service, and then exchange them for corresponding X.509 certificates. No new certificate will be issued when a valid certificate has already been delivered – revoke first.

Each new Authentication Daemon will run as a separate process.

6.9.3. Certificate Creation Steps

1. If a *certificate* (*PKCS#12 file*) is not found at the location referenced in the configuration go to step 3
2. If the certificate is valid go back to step 1.
3. If the *PVACMS service is not available* to create a new certificate, then wait until it is then go to step 1
4. *Get credentials* from configured authentication method.
5. Get the key-pair from the configured PKCS#12 file or create a *key-pair*
6. Create a *certificate creation request* (CCR) containing the public key, the credentials and any authentication-method-specific verification data and pass to PVACMS.
7. For all except **basic-credentials** PVACMS *verifies the credentials* and the verification data, if provided, with the credentials-issuer or data provided by the issuer.
8. PVACMS generates a signed X.509 certificate corresponding to the CCR payload contents if verification succeeds.
9. PVACMS uses PVANOTIFY to *notify all concerned* parties that a new certificate has been issued with the specified credentials, certificate serial number and SKID.
10. New certificate is returned to the requesting AD.
11. The AD creates a PKCS12 file and optionally a root certificate if none has been previously created and stores them in the configured locations
12. Go back to step 1

6.10. Authentication Daemons

Authentication Daemons are small programs that run on an EPICS agent and provide a way in which an EPICS agent can obtain an X.509 certificate.

Each daemon implements specific tasks: **credentials acquisition**, and **CCR creation** which will take place in the requesting daemon, and **verification of the CCR** which will take place in the PVACMS.

The CCR will be a polymorphic object that will contain an optional verifier field of any type that will be able to be used by authentication methods to store any data they require to validate the CCR data.

6.10.1. Authentication Daemon credential verification

- The design supports three types of credentials verification method.
  - TYPE 0: Verification for *Basic-Credentials* (Default/Fallback) – no verification performed – e.g. login-name and hostname
  - TYPE 1: *Independently Verifiable Credentials* e.g. **JWT**
  - TYPE 2: *Credentials Verification requiring integration with Authentication Systems' API* e.g. **Kerberos** and **Kerberos over LDAP**.

> **General Note on Support**: The design implements a pluggable support for authentication methods. Each site can decide which of the bundled authentication support modules it wants to include when building the Secure PVAccess components, by configuration settings. Furthermore, new modules can be added in the future by the repo maintainers.

- TYPE 0: Verification for Basic-Credentials
  - If the user runs the `authnstd` daemon, then certificates will be provisioned by the Basic-Credentials AD.
  - It will determine the basic credentials by getting the logged-on-user (or process-name or value of the `EPICS_PVA_AUTH_PROCESS_NAME` environment variable) and the hostname (or IP-Address, or value of the `EPICS_PVA_AUTH_DEVICE_NAME` environment variable).
  - These credentials are used to obtain a certificate that are passed to and EPICS agent it to authenticate itself to a peer on a Secure PVAccess network.
  - The certificate will be provisioned automatically by the PVACMS if available.
  - The certificate will be saved in a file that is accessible only to the EPICS agent it is issued to.
  - Administrators will be notified whenever a certificate is provisioned so that they can optionally approve its use and grant permissions. All TYPE 0 certificates contain the EPICS custom certificate extension that mandates that they can only be valid if used in conjunction with verified status delivered by the PVACMS.
  - *These credentials are not digitally verified* but are optionally approved by network administrators before they can be used.

- TYPE 1: Independently Verifiable Authentication Tokens
  - These are tokens that can be digitally verified by decoding their signature with a public key obtained from the trusted issuer, or by contacting the trusted issuer to get a key to decode the signature, or by sending the token to the issuer by HTTP or some other method to be validated.
  - Such tokens are bearer tokens, i.e. whoever has access to them can assume the role that is claimed by the token. There is no private key needed to prove identity for these types of tokens. It is very important that these tokens are stored and accessed securely.
  - JWTs are an example of this type of token.
  - **<u>STEPS</u>**
    - First the token needs to be delivered to the JWT AD. The AD will listen on a configured port for a token to be delivered in an `HTTP POST` message to `/token`. The payload will be a token represented as a text string corresponding to the following JSON example:

```
{
  "token":
"eyJhbGciOiJIUzI1NiIsImtpZCI6IjMiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwOi8vMTI3LjAuMC4x
OjUwMDAvdmFsaWRhdGUiLCJraWQiOiIzIiwiZXhwIjoxNzI1OTY1MjExLCJpYXQiOjE3MjU5NjM0MTEsIm5i
ZiI6MTcyNTk2MzQxMSwic3ViIjoiZ2VvcmdlQGVwaWNzLm9yZyIsImF1ZCI6IlNlY3VyZSBQVkFjY2VzcyBK
V1QgVGVzdGluZyJ9.pfLO_GaxwltenRy7a54GovjuXR0sZ6BZZXEI4zsPE6k"
}
```

- The AD will then package the raw token up into a CCR message with the `type` set to `jwt` and send it to PVACMS.
- When the PCACMS receives the request, it will attempt to verify the authenticity of the token with the issuer by decoding the payload.
    - It may extract the issuer's validation uri (`iss`).
    - It will check that the issuer's validation uri is from a trusted issuer.
    - **POST**ing (or **GET**ing) the token to the verification uri will return a message that will indicate the validation status of the token.
- If the token is valid then we will extract the other claims of the token (subject (`sub` – split into `name` and `organization`), `organization_unit` (`aud` audience), `not_before` (`nbf`), and `not_after` (`exp` expiry) to be used in the construction of the certificate.
- The certificate will be delivered to the JWT AD which will create the physical certificate file on disk for the client, or server, which will be available immediately for connections.
- An administrator may setup EPICS security rules in advance to authorize JWT clients based on their usernames.

- TYPE 2: Verification requiring integration with Authentication System's API
    - These are authentication mechanisms that require code integration (in EPICS agents) with their APIs to determine and validate the credentials. For example, Kerberos tickets can only be accessed using Kerberos APIs or higher-level abstraction APIs such as GSS-API. LDAP is a directory service, so it does not have any authentication components, but it is often used in single sign-on scenarios as the user database. In such scenarios it is often paired with an authentication mechanism such as Kerberos and then wrapped in a higher-level abstraction layer such as GSS-API.
    - Once the credentials are ascertained, the only way to verify their authenticity on the PVACMS side before issuing certificates is to contact the issuer, again using proprietary APIs.

> **Note**: In fact, PVACMS will already have been registered as a Kerberos service so it will already have the keys (`keytab`) required to decode and validate GSS-API tokens it receives from the client without contacting the KDC.

- **STEPS**
- On the KRB or LDAP AD
    - First, we get the credentials. Using GSS-API we can get the credentials independently of the underlying authentication method.
    - Then we will use GSS-API to construct a token based on these credentials.
    - This token will be added to the CCR.
- Once the CCR is constructed it will be sent to PVACMS along the normal PVAccess link.
- The PVACMS will decode and check the token
    - Decode the token extracted from the CCR.
    - Using GSS-API we will be able to use it to get the peer credentials.
    - Verify that the details of the credentials (name, and validity) exactly match those requested in the CCR.
    - **For LDAP only**: Connect to LDAP service and verify that distinguished name is found in directory
- Proceed as normal to issue the certificate.

6.10.2. AUTHZ Message

The AUTHZ message is optionally included in the PVAccess connection establishment process. In a Secure PVAccess network this is purely for debugging because all of fields (name, method and authority) are retrieved securely from the TLS context and used for authorization to give access to control system resources.

We will concatenate the **CN** (e.g. greg) and **O** (e.g. SLAC.STANFORD.EDU) from the subject of the peer-certificate to produce the *name* field (e.g. greg@SLAC.STANFORD.EDU, or ed@centOS01, or archiver@192.168.2.10).

We update the field called *method* (the type of connection e.g. **tls** (authenticated name came from TLS peer-certificate), or **ca** (name came from regular Channel Access method - i.e. unauthenticated hard-coded value or command-line parameter))

We also add another field called *authority* (the authority for the information contained in the message e.g. **x509** (for mutually authenticated TLS connections where the client and server have certificates), **ca** (for server-only authenticated connections or for legacy non-tls connections), or **anonymous** (when no credentials are supplied or found)).

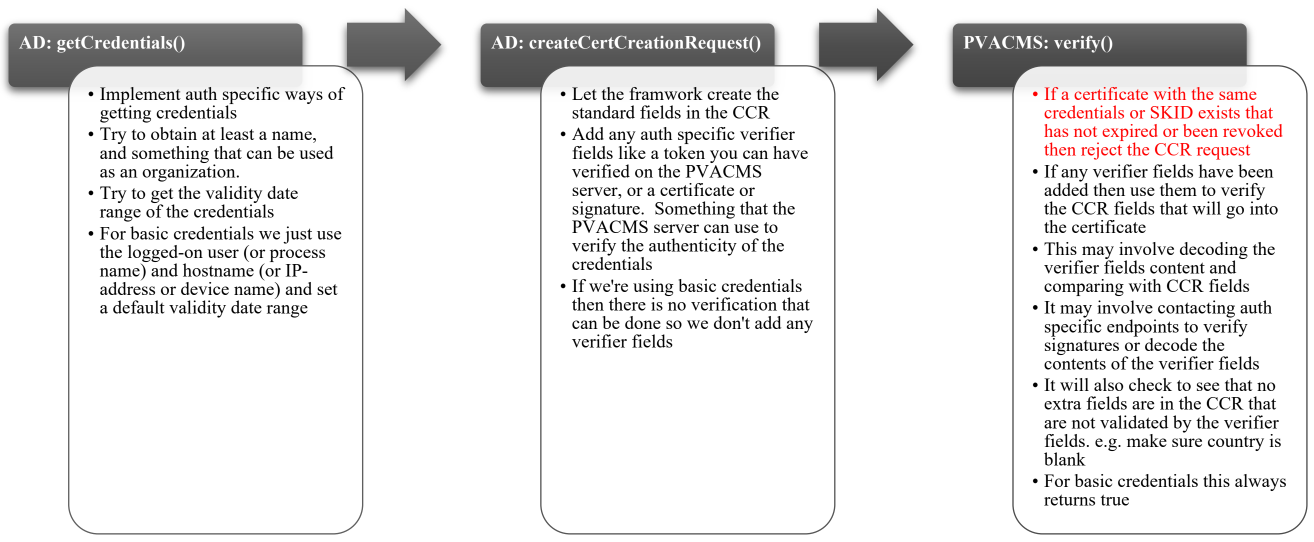
6.10.3. Types of independent verifications supported

For credentials that are independently verifiable three aspects of the verification process are configured to customize the process for each site's specific needs. When calling the verification URI (e.g. `EPICS_AUTH_JWT_TRUSTED_URI`) we configure the following:

1. *Payload Format*: (e.g. `EPICS_AUTH_JWT_REQUEST_FORMAT`) Call endpoint with raw text or formatted payload.
2. *HTTP Method*: (e.g. `EPICS_AUTH_JWT_REQUEST_METHOD`) Call endpoint with **POST** or **GET**
3. *Interpreting Response*: Response is formatted text or a raw value (e.g. `EPICS_AUTH_JWT_RESPONSE_FORMAT`), or uses the HTTP response status – `200` means valid (e.g. if `EPICS_AUTH_JWT_REQUEST_METHOD` is selected.

6.10.4. Authentication method implementation

All Authentication Method classes will be derived from the same base class `Auth`. The subclasses must implement three methods: `getCredentials()`, `createCertCreationRequest()`, and `verify()`. The first two run on the AD and the latter on the PVACMS server. This simple template makes code easy to read, understand, and maintain.



AD: getCredentials()

- Implement auth specific ways of getting credentials
- Try to obtain at least a name, and something that can be used as an organization.
- Try to get the validity date range of the credentials
- For basic credentials we just use the logged-on user (or process name) and hostname (or IP-address or device name) and set a default validity date range

AD: createCertCreationRequest()

- Let the framwork create the standard fields in the CCR
- Add any auth specific verifier fields like a token you can have verified on the PVACMS server, or a certificate or signature. Something that the PVACMS server can use to verify the authenticity of the credentials
- If we're using basic credentials then there is no verification that can be done so we don't add any verifier fields

PVACMS: verify()

- If a certificate with the same credentials or SKID exists that has not expired or been revoked then reject the CCR request
- If any verifier fields have been added then use them to verify the CCR fields that will go into the certificate
- This may involve decoding the verifier fields content and comparing with CCR fields
- It may involve contacting auth specific endpoints to verify signatures or decode the contents of the verifier fields
- It will also check to see that no extra fields are in the CCR that are not validated by the verifier fields. e.g. make sure country is blank
- For basic credentials this always returns true

Each site will select the supported authentication methods with build-time macros. Users will run whichever AD that the site selects and the PVACMS server will support creating certificates of that type:

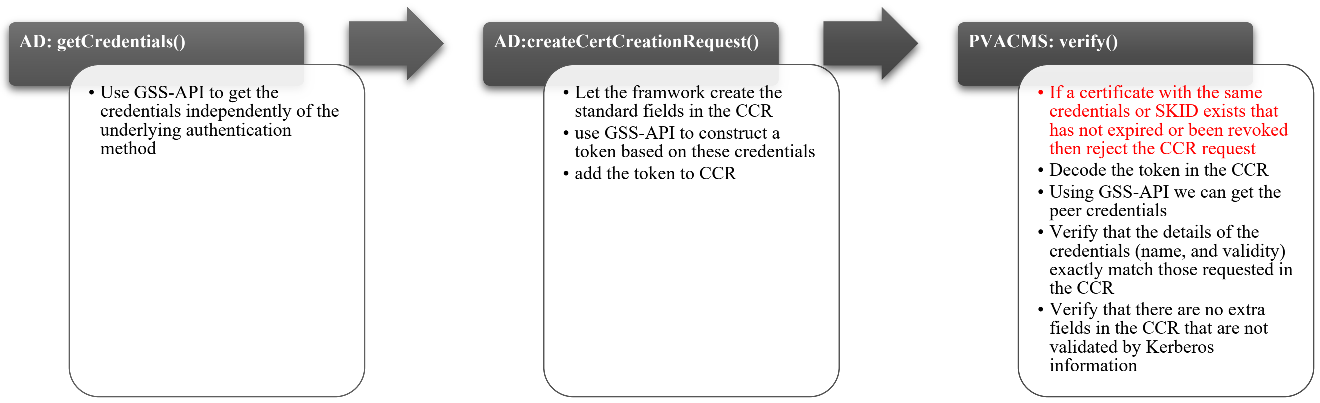
6.10.4.1. Kerberos Service Ticket support

Kerberos is an authentication protocol that uses secret-key cryptography to authenticate clients to services and vice versa. It uses tickets, including Ticket Granting Tickets (TGTs), which are encrypted with keys derived from user passwords or other secrets.

Kerberos tickets can only be accessed using Kerberos APIs or higher-level abstraction APIs such as GSS-API.

Once the credentials are ascertained, the only way to verify their authenticity on the PVACMS side before issuing certificates is to contact the issuer (or use pre-assigned keys from the issuer), again using proprietary APIs.

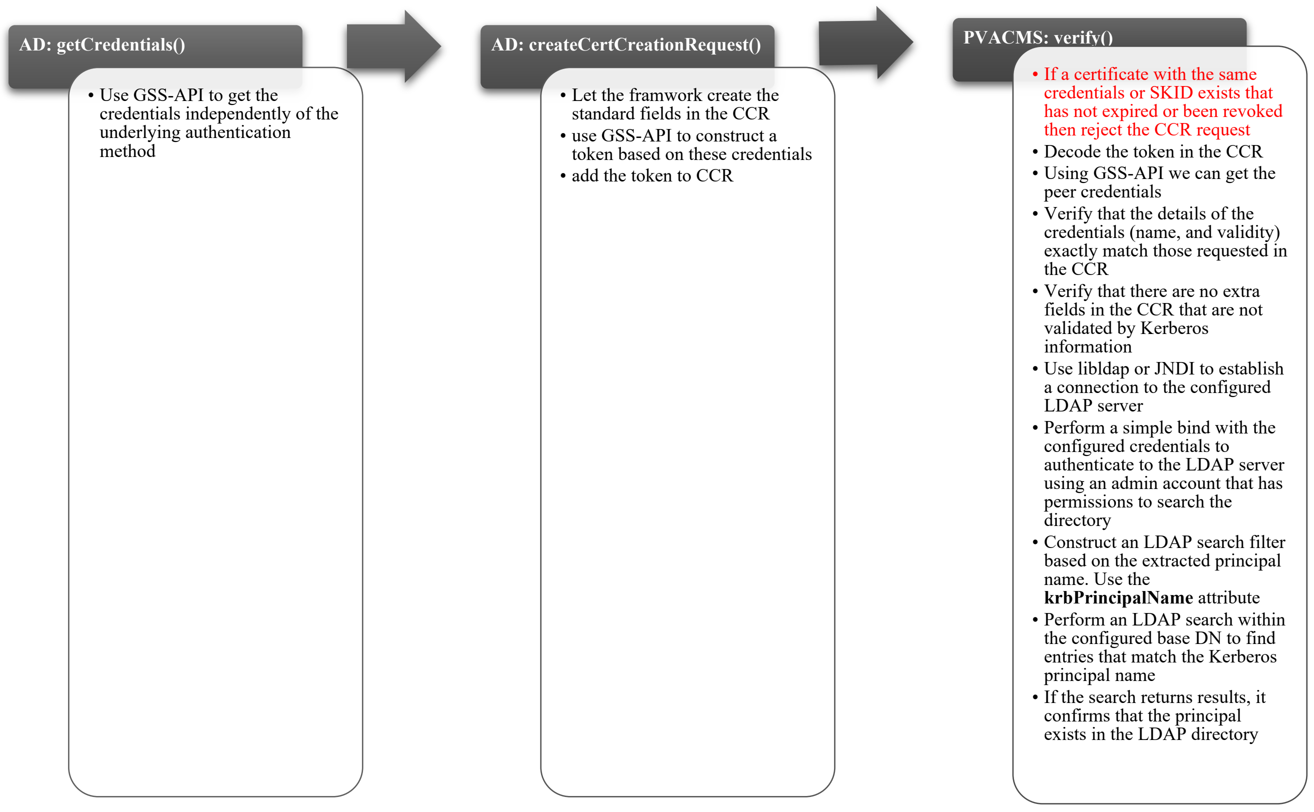
Note: In fact, PVACMS will already have been registered as a Kerberos service so it will already have the keys required to decode and validate GSS-API tokens it receives from the client without contacting the KDC.


AD: getCredentials()
• Use GSS-API to get the credentials independently of the underlying authentication method
AD:createCertCreationRequest()
• Let the framwork create the standard fields in the CCR
• use GSS-API to construct a token based on these credentials
• add the token to CCR
PVACMS: verify()
• If a certificate with the same credentials or SKID exists that has not expired or been revoked then reject the CCR request
• Decode the token in the CCR
• Using GSS-API we can get the peer credentials
• Verify that the details of the credentials (name, and validity) exactly match those requested in the CCR
• Verify that there are no extra fields in the CCR that are not validated by Kerberos information

6.10.4.2. Kerberos on LDAP support

Kerberos and LDAP are two distinct technologies often used together in enterprise environments to enhance security and manage user authentication and authorization. Kerberos is a network authentication protocol designed to provide secure authentication for users and services through a system of tickets and secret keys. LDAP (Lightweight Directory Access Protocol), on the other hand, is a protocol used to access and manage directory information services, typically for storing user information, credentials, and access rights. In a typical setup, Kerberos handles the authentication process, ensuring that users are who they claim to be, while LDAP is used to store user credentials and attributes. Once a user is authenticated by Kerberos, LDAP can then be queried to retrieve the user's information and authorization details, thus providing a comprehensive solution for secure access control and user management.

Support for LDAP is the same as the Kerberos support except we will check to see if the principal comes from the LDAP directory by using `libldap` or `JNDI` in the `verify()` method.



AD: getCredentials()

- Use GSS-API to get the credentials independently of the underlying authentication method

AD: createCertCreationRequest()

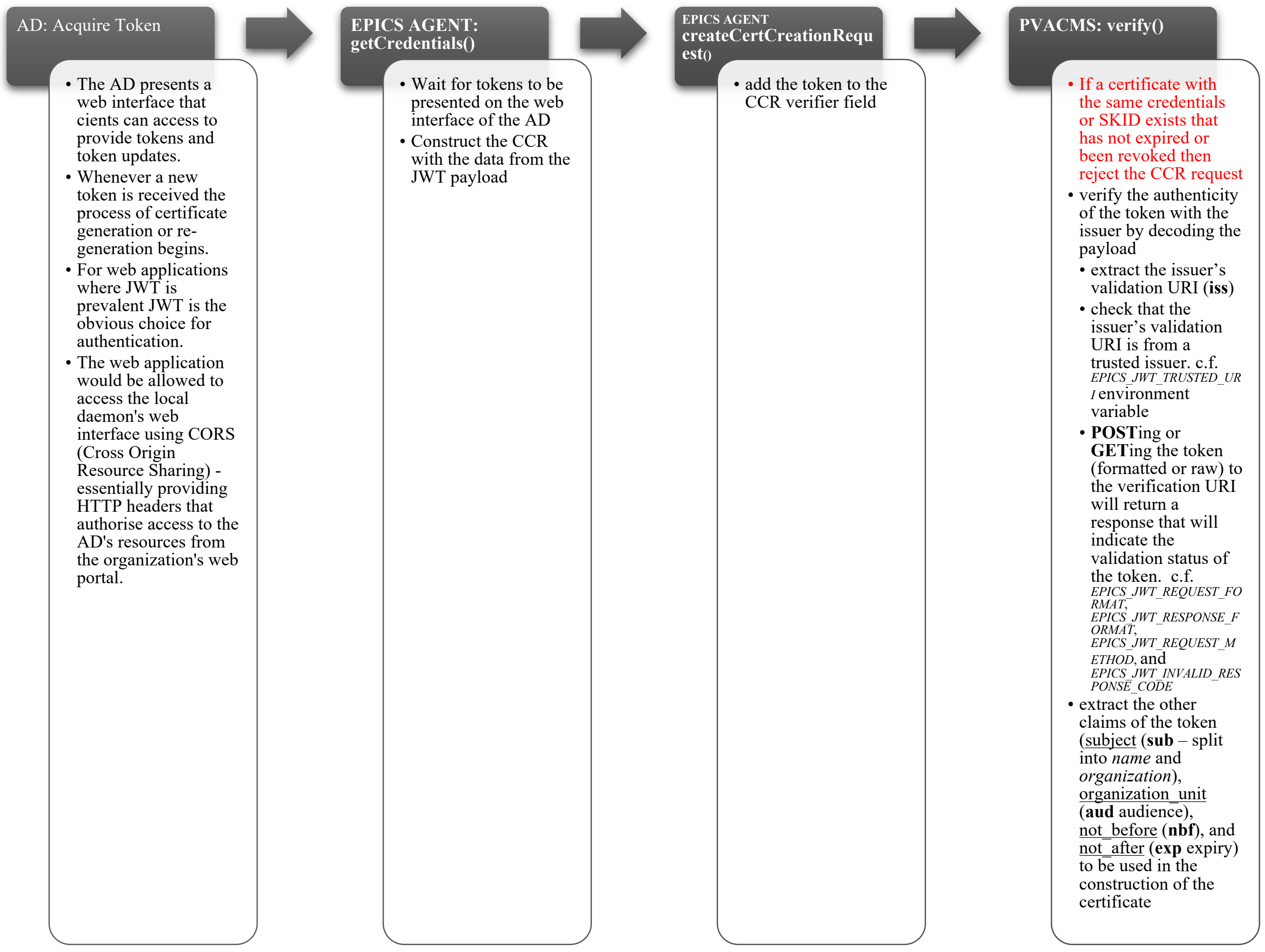
- Let the framwork create the standard fields in the CCR
- use GSS-API to construct a token based on these credentials
- add the token to CCR

PVACMS: verify()

- If a certificate with the same credentials or SKID exists that has not expired or been revoked then reject the CCR request
- Decode the token in the CCR
- Using GSS-API we can get the peer credentials
- Verify that the details of the credentials (name, and validity) exactly match those requested in the CCR
- Verify that there are no extra fields in the CCR that are not validated by Kerberos information
- Use libldap or JNDI to establish a connection to the configured LDAP server
- Perform a simple bind with the configured credentials to authenticate to the LDAP server using an admin account that has permissions to search the directory
- Construct an LDAP search filter based on the extracted principal name. Use the **krbPrincipalName** attribute
- Perform an LDAP search within the configured base DN to find entries that match the Kerberos principal name
- If the search returns results, it confirms that the principal exists in the LDAP directory

6.10.4.3. JWT token and OAuth Support

JWTs are bearer tokens, i.e. whoever has access to them can assume the role that is claimed by the token. There is no private key needed to prove identity for these types of tokens. It is very important that these tokens are stored and accessed securely.



AD: Acquire Token

- The AD presents a web interface that cients can access to provide tokens and token updates.
- Whenever a new token is received the process of certificate generation or re-generation begins.
- For web applications where JWT is prevalent JWT is the obvious choice for authentication.
- The web application would be allowed to access the local daemon's web interface using CORS (Cross Origin Resource Sharing) - essentially providing HTTP headers that authorise access to the AD's resources from the organization's web portal.

EPICS AGENT: getCredentials()

- Wait for tokens to be presented on the web interface of the AD
- Construct the CCR with the data from the JWT payload

EPICS AGENT **createCertCreationRequest()**

- add the token to the CCR verifier field

PVACMS: verify()

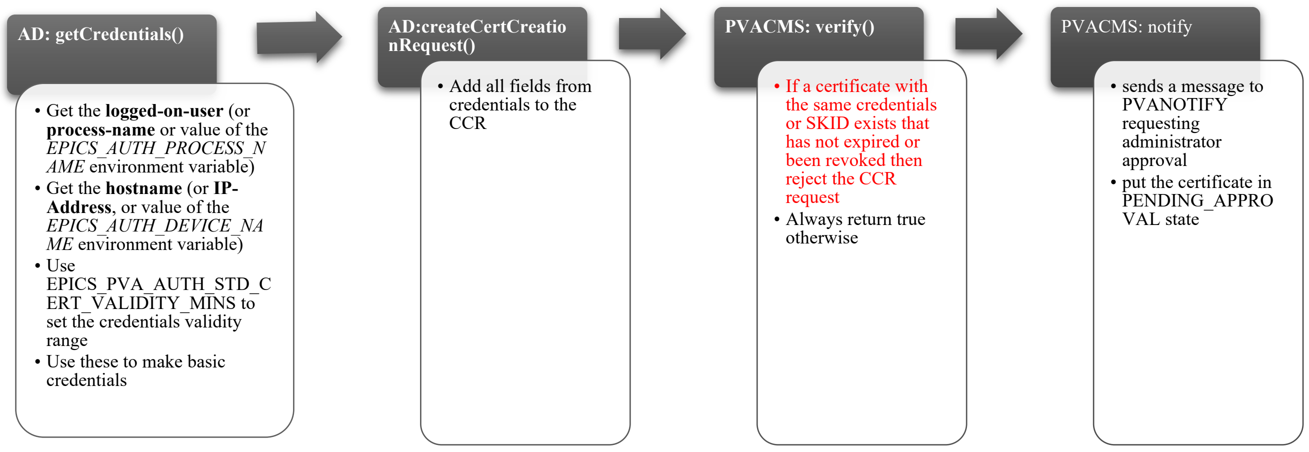
- If a certificate with the same credentials or SKID exists that has not expired or been revoked then reject the CCR request
- verify the authenticity of the token with the issuer by decoding the payload
  - extract the issuer's validation URI (**iss**)
  - check that the issuer's validation URI is from a trusted issuer. c.f. *EPICS_JWT_TRUSTED_URI* environment variable
  - **POST**ing or **GET**ing the token (formatted or raw) to the verification URI will return a response that will indicate the validation status of the token. c.f. *EPICS_JWT_REQUEST_FORMAT*, *EPICS_JWT_RESPONSE_FORMAT*, *EPICS_JWT_REQUEST_METHOD*, and *EPICS_JWT_INVALID_RESPONSE_CODE*
- extract the other claims of the token (subject (**sub** – split into *name* and *organization*), organization_unit (**aud** audience), not_before (**nbf**), and not_after (**exp** expiry) to be used in the construction of the certificate

Use Case:

An organization that provides access to most of its resources through web applications wants to integrate access to its PVAccess Network with single-sign-on (SSO). When users log into the web interface, they want them to also be logged in as that same user in PVAccess. Users simply run the JWT AD on their local machines and when they are logged in to the web portal, they will also have access to their control system applications, that are based on Secure PVAccess, because the certificates required will have been automatically generated with their logged-on credentials in the background. The administrator will configure the ADs such that they allow Cross Origin Resource Sharing from their web application domain and the web application will share the authentication token (JWT) with the JWT AD. Now command line tools, CS Studio, and other Control System applications will all be able to use the X.509 certificates generated for the JWT-identified user.

6.10.4.4. Basic Credentials

Use the basic authentication AD to generate certificates using the logged-in process information (username, hostname, etc.) or configurable details (process name, device name, organization name):



AD: getCredentials()

- Get the **logged-on-user** (or **process-name** or value of the *EPICS_AUTH_PROCESS_NAME* environment variable)
- Get the **hostname** (or **IP-Address**, or value of the *EPICS_AUTH_DEVICE_NAME* environment variable)
- Use EPICS_PVA_AUTH_STD_CERT_VALIDITY_MINS to set the credentials validity range
- Use these to make basic credentials

AD:createCertCreationRequest()

- Add all fields from credentials to the CCR

PVACMS: verify()

- If a certificate with the same credentials or SKID exists that has not expired or been revoked then reject the CCR request
- Always return true otherwise

PVACMS: notify

- sends a message to PVANOTIFY requesting administrator approval
- put the certificate in PENDING_APPROVAL state

6.10.4.4.1. client and server device names

If `EPICS_AUTH_STD_USE_DEVICE_NAME` is configured, then the AD determines the device name from configured `EPICS_AUTH_STD_DEVICE_NAME`.

6.10.4.4.2. client and server process names

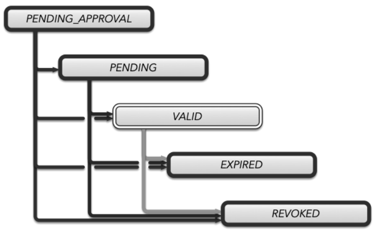
If `AUTH_STD_USE_PROCESS_NAME` is set, then the AD will get the process name and trim off any path component to leave just the base name. If the `EPICS_AUTH_STD_PROCESS_NAME` variable is set, then it will use that instead of the actual process name.

6.11. Operating a Secure PV Access network

6.11.1. TLS operating attributes

- An EPICS agent is TLS-capable if it
  - has a configured X.509 certificate, that is readable and for which a valid password is configured if it is password protected
  - the certificate is valid (not expired or revoked)
  - is compiled with source code that contains the new security features that are the products of this work

- An EPICS agent's certificate state is one of



  - `PENDING_APPROVAL`, `PENDING` – the certificate parses correctly but the PVACMS says that it is waiting for approval to be made `VALID` or its start date is in the future.
  - `VALID` – the certificate parses correctly, has a trusted root certificate, has a valid certificate chain leading to the root certificate, is not expired nor prior to the start date, has not been revoked, has the correct key usage assignments (such as signing, encipherment, etc.), and correctly specifies the subject and other mandatory fields.
  - `EXPIRED` – the certificate has expired
  - `REVOKED` – the certificate has been revoked by network administrators
  - `UNKNOWN` – can't access the PVACMS for a status update and the validity of the last status has expired (stale)

6.11.2.Authentication Mode

- Mutual authentication mode
- Server-only authentication mode
- Un-authenticated mode when credentials are supplied in the `AUTHZ` message but are not verified, as is the case with legacy PVAccess connections
- Unknown authentication mode when no credentials are provided in the `AUTHZ` message.

6.11.3. Operational Exceptions

Lists some common operation conditions that will be handled by the proposed solution

6.11.3.1. Session establishment failures

When the framework fails to establish a session because of a failure at any one of the many steps required, the PVAccess protocol automatically retries 12 times. This can be because of misconfiguration, file input/output problems, loss of connectivity at the TCP or IP layer, rejection due to configured rules, etc.

6.11.3.2. Connection Drops

When connections are dropped at the TCP/IP layer then recovering should be quite easy. The PVAccess Protocol already handles this case by retrying the connection. Retrying the connection will pick up the same configuration and will use the existing certificate to re-establish the TLS session.

6.11.3.3. Expiring certificates during TLS session

In PVAccess many connections are long running, especially connections where a client is monitoring a PV. Certificates will expire during these sessions. The TLS libraries do not automatically detect this anomaly.

We propose two solutions to this use case:

6.11.3.3.1. Monitoring of certificate status

All EPICS agents will monitor the certificate it is using to establish the SSL context. If it expires then the current connections will be dropped and any new connections will fall back to tcp connections.

If an AD is configured and running it may detect the expiration of the certificate and regenerate another one before any new connections are made.

6.11.3.3.2. Long Duration Certificates

The PVACMS can be configured to ignore the expiration times set in the CCR and create certificates that have very long duration. E.g. 10 years. These certificates will always have the custom extension set which contains the PV name to connect to for status updates. These certificates statuses will only be accepted based on the PVACS service certificate status and not by the expiry date encoded in the certificate. The OCSP field `nextUpdate` is used to determine the shorter-term expiration periods that would normally be encoded into the certificates (a week, a day, 30 minutes). At which time the EPICS agent will be required to contact the PVACS service for an updated status.

6.11.3.3.3. Proposal to ignore Long Duration Certificate's expiration date

There is a proposal to ignore the Long Duration Certificate's expiration date. This would work in conjunction with the AD. When the certificate expires but before the end of the last certificate status validity period the AD would re-request another certificate from PVACMS which would be installed in the EPICS agent but not used to recreate a new SSL context for existing connections. It would instead be checked to see if it has the same SKID as the current certificate and if it does then the PVACS service could be contacted to give status on that certificate and the result applied to the expired certificate – thus extending its life indefinitely. Ignoring a certificates expiration date represents a significant departure from the norms and we will need to discuss this with the community before we adopt such a radical approach.

6.11.3.4. Revoked certificates during a TLS session.

In PVAccess many connections are long running, especially connections where a client is monitoring a PV. Certificates may be revoked during these sessions. The TLS libraries are completely unaware of this state. In our proposed solution we will monitor the certificate status using the PVACS service for the certificate an EPICS agent is using and will immediately drop a connection if a certificate it is relying on is revoked. The AD will try to request another certificate in this case. If this succeeds because the new credentials are acceptable, then the new certificate is installed and new SSL contexts for new connections can be created.

6.11.4. Diskless Server bootup with TLS

Very often EPICS network devices (usually IOCs) boot up without disks. This presents a problem for Secure PVAccess Servers embedded within these IOCs that require access to a certificate but have no disk from which to read it.

With the functionality proposed for the PVACMS to be able to create and provision certificates on the fly, we can have the diskless servers installed with accompanying ADs that will simply try to create a server certificate automatically, requesting one from the PVACMS. A remote disk will have been mounted and we can use it to store the certificate securely and configure the IOC to use it.

- Ensure that there is a network mount available via NFS, SMB/CIFS (Server Message Block/Common Internet File System commonly used by Windows systems), or AFP (Apple Filing Protocol primarily used for Apple devices), among other network filesystem software.
- The configuration will point to a location on the network share that is protected such that only the process can access it.
- It can optionally be further protected by use of a password configured in the diskless server.
- If no PKCS#12 file exists at the referenced location, then the AD will automatically make a CCR request to PVACMS and obtain a certificate which it will install at that location.

The trust relationship will have to have been established in advance during the bootup by installing the root certificate in the device trust store or by using publicly signed root certificates in PVACMS.

6.12. Certificate Management Features

This topic concerns the creation of a Certificate Authority (CA), creation of certificates initiated by ADs, distribution of certificates to EPICS agents, and configuration of EPICS agents to reference the newly delivered certificates.

6.12.1. Certificate Authority

Only implementation in c++ is required as this component will exist only as a c++ executable - `pvacms`.

A certificate authority (CA) is an entity responsible for issuing, managing, and revoking digital certificates. These certificates are used to verify the identity of EPICS agents in our Secure PVAccess network, ensuring secure communication and authentication over internal and external networks such as the internet.

Normally sites will create their own CA, but it is advisable that the root certificate for any site be signed by a public certificate authority.

In PVACMS this can be done automatically by simply starting the first PVACMS node. When the first node is started, if a CA is not found at the location pointed to by *`EPICS_PVACMS_CA_KEYCHAIN`* then one is created there with the parameters specified on the command line or in the environment variables for that purpose.

6.12.2. Automatically provision certificates when missing or invalid

We have already talked broadly about how to automatically provision certificates in many scenarios. Here we'll just describe some of the concrete details of the CCR interface.

- When an AD starts up it will look for keychain file and if it doesn't exist or contains an invalid certificate then it will try to generate a new one automatically.
  - *Look for or generate a key pair* – If a key pair file exists use that one otherwise creating a key pair involves initializing the OpenSSL context and generating the keys.
    - Use `EVP_PKEY_CTX` to create the context.
    - Use `EVP_PKEY` to store the keys.
  - *CCR creation* – The CCR is a PVAccess structure that has the following structure.

```
Structure
    string     type
    string     name
    string     country        : optional
    string     organization   : optional
    string     organization_unit    : optional
    string     for_client
    string     for_server
    UInt32     not_before
    UInt32     not_after
    string     pub_key
    structure  verifier
       verifier_fields        : optional
```

- *CCR transmission* – the CCR is sent to the `CERT:CREATE` Secure PVAccess endpoint. A flag on the connection called `server-only` is used to mandate that only the server side of the connection be authenticated (to avoid infinite recursion). The call is an RPC call with the single query parameter called **ccr**.
- *CCR processing* – On the PVACMS server the CCR is processed as described earlier
- *Certificate return value* – if the CCR is verified (for TYPE 1 or TYPE 2 requests) then the certificate is generated with the correct attributes and is packaged as a PEM string and sent back to the client over PVAccess. `PEM` (Privacy-Enhanced Mail) is a simple string format used for representing certificates, public keys, and other cryptographic data. It encodes the binary data in Base64 and includes header and footer lines for easy identification. This is ideal for transferring the certificate back to the requesting EPICS agent. Multiple elements can be included in the same string with a simple concatenation. We include the certificate and its public key, any certificates in its certificate chain (in order), and the root certificate.

```
-----BEGIN CERTIFICATE-----
MIID...<base64-encoded certificate data>...FJ8=
-----END CERTIFICATE-----
-----BEGIN PUBLIC KEY-----
MFww...<base64-encoded public key data>...IDAQAB
-----END PUBLIC KEY-----
```

The returned value from the RPC encapsulates this PEM data in the following structure in the **cert** field:

```
structure
    UInt64     serial
    string     issuer
    string     cert
    alarm_t    alarm
```

6.12.3. Revoke certificates

We have already talked broadly about how to revoke certificates. Here we'll just describe the concrete details.

- **PUT** `CERT:STATUS:issuer_id:serial_number status=REVOKED` to revoke a certificate.
- Must be on a mutually authenticated connection
- User must be referenced in the UAG of the PVACMS as an administrator
- Use the PVAccess PUT message type
- The PV name references the certificate that needs to be updated.
  - `issuer_id` – this is the hash of the certificate issuer's public key represented aas hex digits and truncated to 8 characters
  - `serial_number` – this is the serial number of the certificate who's status is requested

When this is received by a PVACMS node it will mark the certificate that it references as `REVOKED` once it has checked the credentials of the caller.

6.12.4. Create PKCS#12 file and Root Certificate

We have already talked broadly about how these files are created. Here we'll just describe the concrete details.

- *Creation of the PKCS#12 file* is done by combining the following items:
  - The Certificate from the PEM string
  - The Certificate chain from the PEM string (don't include the root certificate, just the reference to it)
  - The private key that was not sent to PVACMS but was kept in the AD.
- *Creating the Root Certificate file* – this is created from the last certificate in the certificate chain in the PEM string. It is written out directly as a PEM file. With every distribution of the SSL libraries there's a well-known place where root

certificates are stored on disk and this location can be queried via the API. We use this location to store the root certificate. This is not the trust store. There's nothing private in the root certificate so it's not necessary to keep it secure.

- *Trusting the Root Certificate*. If we don't find the root certificate already in that location (and it's not a public certificate authority's root cert) then we will print a message on the AD console of how to trust that root certificate and show its location.

6.12.5. PVACS: Certificate Status over PVAccess

The PVAccess protocol refers to the resources it controls using PV names. These PV names uniquely identify a resource within a site. We introduce, with this project, PV Names for accessing and controlling X.509 certificates.

An X.509 certificate is identified by a an issuer id and a serial number.

GET `CERT:STATUS:issuer_id:serial_number`

This will get the status of the referenced certificate. It returns a timestamped certificate status object as follows:

```
Structure
    enum_t     status
    UInt64     serial
    string     state
    enum_t     ocsp_status
    string     ocsp_state
    string     ocsp_date
    string     ocsp_certified_until
    string     ocsp_revocation_date
    UInt8A     ocsp_response
```

The `ocsp_response` field is the only field that is used by the EPICS agents to determine certificate status. All other fields are convenience fields. The `ocsp_response` field contains the signed and certified certificate status that is comprised in the other fields. This alone can be trusted by the EPICS agent which will decode and verify the field before accepting the status.

6.12.6. PVACS stapling integration with TLS handshake

There is also a TLS handshake extension called "OCSP Stapling", also known as the TLS Certificate Status Request extension. OCSP Stapling is designed to improve the efficiency of the OCSP by reducing the need for a separate round trip from the client to the OCSP responder.

- *PVACS stapling*: Use it on the server side during a TLS handshake to get the certified server certificate status and send it along with the certificate so that the client can avoid the verification step. Set the OCSP callback in the server and client to implement this functionality.

  Here's how OCSP Stapling works:

OCSP Stapling is defined in `RFC 6961 [21]` and is an extension to the standard OCSP. During the TLS handshake, the Secure PVAccess client indicates support for OCSP Stapling by including the `"status_request"` extension in the *`ClientHello`* message. The Secure PVAccess server will include a `"CertificateStatus"` message in the server's response, which contains the stapled OCSP response in PKCS#7 format (the same format returned in `ocsp_response` by the PVACS service).

- As we are already connected to the Secure PVAccess network we simply have the server `GET` the current certificate status, timestamped and signed by the PVACMS (CA) which is trusted by the clients in any case. Here's how we do it (example shown for openssl c++ implementation):
  - On the server-side:
    - Start off the TLS handshake process as usual.
    - When preparing the *`ServerHello`* message, create a callback using the **`SSL_CTX_set_tlsext_status_cb()`** function to generate the custom OCSP response data.
    - Inside the callback function, call **GET** on **`CERTS:`***`{issuer_id}`***`:`***`{serial_number}`* where *`serial_number`* is the serial number of the certificate, and *`issuer_id`* is the issuer id of the certificate issuer, to retrieve the signed certificate status. Use the returned *`ocsp_response`* byte array.
    - Allocate memory and copy the bytes into this newly allocated memory location.
    - Use **`SSL_set_tlsext_status_ocsp_resp()`** to set this data as the OCSP response for the handshake.
    - Now continue the usual TLS handshake process.
  - On the client-side:
    - Start the TLS handshake process as usual.
    - Set callback function using **`SSL_CTX_set_tlsext_status_cb()`** to process the custom OCSP response during the handshake.
    - In the callback first extract the *`serial`* number, *`CN`*, *`O`*, *`OU`*, *`C`*, *`notBefore`*, *`notAfter`* from the certificate
    - Extract the OCSP response using **`SSL_get_tlsext_status_ocsp_res()`**
    - Decode the blob into *`status_date`* and *`signature`*. Then decode the signature using the public key of PVACMS and see that the decoded text contains the same values as *`serial`* number, *`CN`*, *`O`*, *`OU`*, *`C`*, *`notBefore`*, *`notAfter`* and *`status_date`*.
    - If all checks pass, allow the handshake to proceed as usual

6.12.7. PVACS integration with PVAccess server for own-certificate verification

The PVACS service of the PVACMS can provide information about the status of any certificates under management. The Secure PVAccess server can MONITOR this service to check its own certificate status and can get a signed assertion from the PVACS service to attest to the certificate's validity that it can include in the TLS handshake so that the client doesn't have to check the validity with an extra network round trip. The latter process is known as OCSP stapling.

The Secure PVAccess server will GET the current status during the TLS handshake of each new connection and staple the result to the handshake.

6.13. Notification System

6.14. Supporting non-EPICS network clients

6.14.1. Add OCSP-PVA service.

- Add the OCSP-PVA service to provide OCSP services to network clients that don't speak PVAccess.

6.15. Gateways

The PVAccess gateway will benefit from the C++ source changes in PVXS. The PVXS server and client components are already reused in the PVAccess gateway so whatever changes we make to implement the Secure PVAccess Protocol will be automatically available in the PVAccess Gateway.

6.16.Tool updates

All tools will be updated as a result of the updates to the underlying PVAccess protocol.

6.16.1. softIocPVX (qsrv)

This is the command-line interface that allows executing a softIOC.

We propose adding a reconfigure command to the shell to force it to re-read certificates in case they change.

6.16.2. pvacms

This is a new component that provides a PVACMS cluster node functionality.

```
Usage: pvacms -a <acf> <opts>
  -a <acf>            Access Security configuration file
  -h                  Show this message.
  -V                  Print version and exit.
  -v                  Make more noise.
  -k <keychain file>  Specify keychain file location
                      Overrides EPICS_PVACMS_TLS_KEYCHAIN
                      environment variable.
                      Default ca.p12
                      append ;password for password
 -d <cert db file>    Specify keychain file location
                      Overrides EPICS_PVACMS_TLS_KEYCHAIN
                      environment variable.
                      Default ca.p12
                      append ;password for password
 -n <ca_name>         To specify the CA's name if we need
                      to create a root certificate.
                      Defaults to the PVACMS
 -o <ca_org>          To specify the CA's organization if we need
                      to create a root certificate.
                      Defaults to the hostname.
                      Use '-' to leave unset.
```

6.16.3. odbc-pva

This is the new OCSP service for non PVAccess network clients that leverage certificates managed by PVACMS

```
Usage: ocsppva <opts>

  -h                  Show this message.
  -V                  Print version and exit.
  -v                  Make more noise.
  -p <port>           Specify port to listen on
```

7. Source, Building, and Releases

7.1. Build Macros

```
PVXS_ENABLE_KERBEROS_AUTH  = NO
PVXS_ENABLE_JWT_AUTH  = YES
PVXS_ENABLE_LDAP_AUTH  = NO
```

Note: This applies to c++ builds only.

- These build time macros will be added to a local `CONFIG_SITE.local` file placed at one directory above the root of the distribution and will be picked up if it exists:
  - *`PVXS_ENABLE_JWT_AUTH`*: If this is set then the executables and libraries will be built with JWT support such that the new JWT AD will be built.
  - *`PVXS_ENABLE_KERBEROS_AUTH`*: If this is set then the executables and libraries will be built with JWT support such that new KRB AD will be built.
  - *`PVXS_ENABLE_LDAP_AUTH`*: If this is set then the executables and libraries will be built with JWT support such that new LDAP AD will be built.
  - ***`NOTE:`*** *`Basic credentials`*: By default, support for basic credentials AD (`authnstd`) is always included.
  - *`PVXS_ENABLE_SSLKEYLOGFILE`*: This controls whether the executables are built with SSL key logging enabled. When enabled ssl keys are sent to the specified file and so a network packet scanner like Wireshark can use these keys to decode the encoded TLS packets. This must be used only for testing and never in production environments.

7.2. Source Control

- *Location*: Currently the C++ code is developed in a private repository of Michael Davidsaver [19]. We will move the code into the main EPICS core repository when we get the go-ahead from the EPICS steering committee.

  The Java code is developed and integrated into the Phoebus repository [20]. We would like this to be moved to the EPICS PVA Java core repository but we don't have a roadmap for when that will occur.

- *Branches*: All code will be developed on a separate branch and delivered into the *`main`*/*`master`* branch as part of a normal release process. Code will not be developed directly into the master branch. This is to avoid any bugs, and operational degradation that may occur with releasing untested code in this way. We will constantly rebase the branches up until release to make sure that they are easy to merge, and so that we don't miss any significant changes in the master.

  Unfortunately, prototype code for the Java implementation was made in the Phoebus master branch. This code is therefore publicly available and is compiled into every new Phoebus build. We will attempt to separate out this code and follow the branch strategy laid out above.

Repos: The development environment will encompass, mainly, two existing repositories:

- The **PVXS repo** (https://github.com/mdavidsaver/pvxs) which houses C++ code for EPICS agents and includes multiplatform and multi-architecture support, requires the **epics-base** repo (https://github.com/epics-base/epics-base) to be built.
- **Phoebus repo** (https://github.com/ControlSystemStudio/phoebus), a component of Control System Studio, that contains Java client libraries.
- While the PVXS repo is currently private, plans are in motion for it to replace the **pvAccessCPP repo** (https://github.com/epics-base/pvAccessCPP).
- The Phoebus repo is part of the larger CS Studio project but will be separated out to replace existing code in **pvAccessJava** (https://github.com/epics-base/epicsCoreJava/tree/master/pvAccessJava) which is a subset of **epicsCoreJava** (https://github.com/epics-base/epicsCoreJava).
- We will finish the development before relocating to new repositories. This approach guarantees minimal disruption to ongoing work and minimizes potential configuration and testing problems during the development phase.

Branches: Currently, the branch structure in the PVXS repo flows from **master** to **tls**, and then to **tls-std**. This needs to be refactored so that **tls-std** branches directly from **master**. The changes already committed in the **master** branch of the Phoebus repo should be migrated to a dedicated **tls** branch, ensuring that master remains the primary branch for stable operations.

Project Management:

- Merges will be delayed until the end of the first year of development, which would be the end of September 2024. This timing allows for comprehensive testing before the changes are merged into the main branches and would enable broader testing rollouts to start in Q4 2024.
- Delaying further risks too much divergence of the code from master and will lead to significant conflicts when merging.
- Merging sooner will not provide enough time to complete testing.
- The changes being made are mostly protected by conditional compilation guards so only users who wish to take part in the testing need try the new features.

7.3. TLS external libraries

The C++ implementation relies on `OpenSSL` libraries and source, whereas the Java implementation relies on built-in classes from the `javax.net.ssl` package.

- *Build now requires manual installation of OpenSSL binaries and source*. Developer needs to install OpenSSL locally when building PVXS.

7.4. New External Libraries

Note: This applies to c++ builds only.

- GSS-API

The Generic Security Services Application Program Interface (GSS-API) is a standardized interface that provides security services for applications, enabling them to authenticate and establish secure communication channels. GSS-API abstracts the underlying security mechanisms, allowing applications to use various authentication methods, including Kerberos, without needing to be tightly coupled to them. We leverage GSS-APIs for Kerberos, to access Kerberos principals. The GSS-API can initiate and accept security contexts, by exchanging tokens that carry authentication data. By doing so, it verifies the identity of EPICS agent processes based on their Kerberos credentials. This relies on the trusted Kerberos Key Distribution Center (KDC) to validate the principals involved.

- krb5

The Kerberos 5 (krb5) protocol, is delivered as part of the Generic Security Services Application Program Interface (GSS-API). It is one of the mechs provided by GSS-API and as such allows GSS-API to interact with the Kerberos API and KDC.

- jwt-cpp

jwt-cpp is a header-only C++ library designed for creating and verifying JSON Web Tokens (JWTs). It provides a straightforward and flexible API for handling JWTs, enabling developers to securely encode, decode, and validate JWT tokens within C++ applications. With jwt-cpp, you can verify credentials delivered in a JWT token, jwt-cpp allows you to parse the token, extract the claims, and validate the signature against a known secret or public key. Being header-only, it simplifies integration, and we will add it as a sub-module in our project to facilitate JWT handling without requiring additional build steps.

- libevent with openssl.

libevent is a library that provides asynchronous event notification, facilitating efficient network communication by handling multiple simultaneous connections. When combined with OpenSSL, libevent can manage secure TLS connections, leveraging OpenSSL's robust cryptographic functions to ensure data confidentiality and integrity. While libevent can be built with optional support for OpenSSL, this requires that OpenSSL is installed on the system or built from source. In this work cycle, we are integrating OpenSSL version 3 into our project. This integration is crucial for our TLS implementation, as it enables secure communication channels within our asynchronous event-driven architecture. By incorporating OpenSSL, we enhance the security and reliability of our network interactions, ensuring that sensitive data is protected during transmission

7.5. Add new unit tests

Unit tests can be run manually and are also run automatically in ci builds as part of the cross platform/arch compatibility and acceptance testing on each build and before each release.

- TLS tests for EPICS agents.
- Kerberos authentication tests
- Kerberos over LDAP authentication tests
- JWT authentication tests
- Tests for PVACMS certificate management and automatic certificate generation
- Tests for PVACMS cluster operation
- Tests for OCSP-PVA for non PVAccess network clients using PVACMS managed certificates

7.6. Releases, versions, and roadmap

- Release Proposal:
  - We propose release 1.2.3 of PVXS for first Secure EPICS release containing TLS additions, Secure EPICS Authentication extensions, and Secure EPICS PVACMS
- Merging of PVXS into base – this is expected within the current year.
- The current version of the prototype of PVXS is available on the `tls` branch Michael Davidsaver’s git repository [22] and on the `master` branch of the `CS-Studio Phoebus git repository [20]`.

7.6.1. Update PVXS version

- Note: This applies to c++ builds only. Update to 1.2.3.

7.7. Licensing, releases, versions, and roadmap

- *Implementations*:
  - For C++ `PVXS` [19] there is the legacy `pvAccessCPP/pvDataCPP`. [23]
  - For Java `core-pva` [20] there's the legacy `pvAccessJava` and `pvDataJava` in the `epicsCoreJava` [24] repository.

  The legacy implementations remain and receive bug fixes, as well new planned releases for the C++ legacy implementation. The new implementations see additions like IPv6, and Cyber Security, etc.
- *Usage*:
  - The legacy C++ implementation has widespread usage:
    - `QSRV` in IOCs
    - Many python scripts, and
    - the `PVA Gateway`.

    So, there we need to think about updating the IOC to using PVXS. This has been suggested to the EPICS steering committee, and they agree, but have not yet set a data. The PVA Gateway can already be compiled with PVXS, and the python scripts should be able to use PVXS already. Incorporating PVXS into the main EPICS code line would be the optimal solution.
  - For the Java implementations:
    - The predominant user is ***CS-Studio***, where the current implementation (***Phoebus***) includes `core-pva`, so it has the IPv6 support as well as the Cyber Security prototype.
    - Legacy `pvAccessJava/pvDataJava` does not have a lot of usage. The older, Eclipse-based ***CS-Studio***, still uses it. The Archive Appliance uses it, but is in the process of updating to `core-pva`.

7.7.1. Licenses

The license model is currently BSD-3 [25] – The 3-Clause BSD License; but SLAC would like its standard license to be included in the source code.

As these sources are created by several other authors, copyright can't reasonably be attributed solely to SLAC. Furthermore, it doesn't appear that SLAC or Stanford are among the list of recognized Open-Source Initiative (OSI) licenses [26].

In recognition of SLAC's substantial contribution we will add the following to our BSD-3 text in the LICENSE text file.

```
    Copyright (c) 2024 The Regents of the University of California, and The Board of Trustees of the
Leland Stanford Junior University, through SLAC National Accelerator Laboratory (subject to receipt of any
required approvals from the U.S. Dept. of Energy). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided
that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and
   the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions
   and the following disclaimer in the documentation and/or other materials provided with the
   distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or
   promote products derived from this software without specific prior written permission.
```

8. Performance Goals and Testing

8.1. Perf goals

During performance testing, test scenarios will be developed to study the impact under various conditions. At that time, we will be able to finalize performance goals in line with general goals specified below:

- Network Bandwidth/Load increase:
  - For initial certificate provisioning: As there is no provisioning of certificates in PVAccess, we can only give a value per node not an increase.
    - **Bandwidth used be < 100Kb per node**.
    - **Load must be < 10kB per node**
  - General operation: This includes the overhead for encrypting the messages and the initial TLS handshake:
    - **TLS handshake < 15KB**
    - **TLS encryption < 15% increase in bandwidth.**
  - Monitoring status: Will only send messages close to expiration so very low
    - **Bandwidth increase < 10KB per node per year.**

- Increase in time to evaluate ACF function:
  - There will be no change in time to evaluate the ACF function for authorization.

- Connection Establishment Time increase:
  - Connection time increase without certificate provisioning:
    - < 0.1% increase in connection time
    - < 1ms increase in connection time.
  - Connection time increase with certificate provisioning
    - < 5% increase in connection time (including network time)
    - < 2ms increase in connection time (including network time)

Data Retrieval Time:

- Increase in time taken to retrieve data from remote service.
  - < 5% increase in data get time (including network time)
  - < 2ms increase in data get time (including network time)

8.2. Rollout

- *Performance and Operational review*: The plan is to select three U.S. EPICS facilities and work with them to test the system in various types of production environment.

  We will choose sites that can help us test:

  - Performance and administrative challenges with very large numbers of PVAccess servers,
  - Operation with diskless PVAccess servers,
  - Operation with PVAccess gateways,
  - Sites with various authentication methods.

    SLAC will automatically be on the list.

    It is important to note that the selected facilities should ideally have a development environment available where we can run tests that simulate production activity, or have time where the facility is offline or being provisioned such that running tests won't have adverse effects.

    We have had interest from Fermilab and ORNL for help in this testing. ISIS in the UK has expressed an interest in helping us test and in providing help developing the platform but as they are non-US they are offering to self-fund their efforts.

- *Implement Findings*: Once the testing is complete, we will write a report and use that report to implement changes required to mitigate the identified problems.
- *General Availability*: Once these changes have been implemented and the results rechecked in the selected facilities, we will make a formal release of the software so that it is available for general use in all EPICS facilities.

9. Works Cited

[G. McIntyre, G. White, M. Davidsaver, B. Dalesio and K. Kasemir, "EPICS Security Key Decisions impacting Design and Implementation," 2024.

[E. U.S. Govt., "Memorandum M-22-09, Moving the U.S. Government Toward Zero Trust Cybersecurity Principles," 26 January 2022. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf.

[M. Davidsaver, "Secure PV Access Protocol Proposal," 31 August 2021. [Online]. Available: https://slac.sharepoint.com/:w:/s/RemoteOperationsNetworkPerformanceImprovement/Eb6efwh-SQdAhH2xUnikMDkBSOQh_1IBK52WdGgiaJ0XSw?e=pBP2eD.

[G. White, G. McIntyre, A. Alavi, K. Brobeck, A. Chabot, U. Chu, A. Deshmukh, M. Foster, S. Hasan, P. Pandey, A. Perazzo, M. Zelazny and M. DavidSaver, "SLAC Accelerator Computing Cyber Security Summary," 28 September 2022. [Online]. Available: https://slac.sharepoint.com/:b:/r/sites/RemoteOperationsNetworkPerformanceImprovement/Shared%20Documents/Accelerator%20Cyber%20Security/AccelCyberSecurityAssessmentSummary_280802r3.pdf?csf=1&web=1&e=TVIxkw.

[G. McIntyre, "Provision of transport level security (TLS) to the PV Access protocol of EPICS 7," 19 May 2022. [Online]. Available: https://slac.sharepoint.com/:b:/r/sites/RemoteOperationsNetworkPerformanceImprovement/Shared%20Documents/Protocol%20Security/Proposal%20PVA-Security.pdf?csf=1&web=1&e=2pnzrM.

[G. White and M. Hahnert, "SLAC Executive Order 14028 Cybersecurity Project - Internal DOE Funding Request," 8 September 2023. [Online]. Available: https://slac.sharepoint.com/:w:/r/sites/RemoteOperationsNetworkPerformanceImprovement/Shared%20Documents/Protocol Security/DOE Cyber EO Funding FY 24 Science SLAC Osprey v4r2.docx?d=w0cc99cc6502c4a28b28773f2fa8da63a&csf=1&web=1&e=zo9pRY.

[D. Cooper, S. Santesson, S. Farrel, S. Boeyen, E. Housley and W. Polk, "RFC 5280," May 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280.

[M. Jones, Microsoft, J. Bradley, P. Identity, N. Sakimura and NRI, "RFC 7519, JSON Web Token (JWT)," May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519.

[E. K. Moriaty, "RFC 7292, PKCS#12," July 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7292.

[R. Housley, "RFC 5652," September 2009. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5652.

[S. Rose, O. Borchert, S. Mitchell and S. Connelly, "NIST 800-207," August 2020. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf.

[R. Freter and D. Team, "Department of Defence (DoD) Zero Trust Reference Architecture," July 2022. [Online]. Available: https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v2.0(U)_Sep22.pdf.

[CISA, "Zero Trust Maturity Model," April 2023. [Online]. Available: https://www.cisa.gov/sites/default/files/2023-04/CISA_Zero_Trust_Maturity_Model_Version_2_508c.pdf.

[The Open Group, "Zero Trust Commandments," January 2024. [Online]. Available: https://pubs.opengroup.org/security/zero-trust-commandments/.

[D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, "RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," May 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280.

[C. Ellison, "RFC 2692, SPKI Requirements," September 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2692/.

[C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas and T. Ylonen, "RFC 2693, SPKI Certificate Theory," September 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2693/.

[J. Callas, L. Donnerhake, H. Finney and R. Thayer, "RFC 4880, OpenPGP Message Format," November 2007. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2440.

[M. Davidsaver, G. McIntyre, K. Vodopivec, B. Martins, H. A. Silva, T. Ives, S. Rose, P. Milne, P. Karlos, É. N. Rolim and A. Wells, "PVXS github source," Osprey DCS, 14 December 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs.

[K. Kasemir, K. Shroff, G. Weiss, A. Wolk, P. Ch, T. Viash, K. Löki, J. Garrahan, G. Jhang, K. Saintin, S. Brewer, cjenkscybercom, es-sns121 and M. Hu, "Control System Studio - Phoebus github source," 10 January 2024. [Online]. Available: https://github.com/ControlSystemStudio/phoebus.

[Y. Pettersen, "RFC 6961, The Transport Layer Security (TLS), Multiple Certificate Status Request Extension," June 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6961.

[G. McIntyre and M.Davidsaver, "Github PVXS/tls," 14 December 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/tree/tls.

[A. Johnson, M. Davidsaver, M. Sekoranja, M. Kramer, R. Lange, D. Hickin, D. Zimoch, F. Akeroyd, J. Vasquez, S. Veseli, H. Junkes, B. Hill, X. Wang, B. Martins and G. White, "epics-base: pvAccessCPP git repository," 15 December 2023. [Online]. Available: https://github.com/epics-base/pvAccessCPP.

[K. Kasemir, G. Carcassi, K. Shroff, M. Kraimer, D. Hickin, R. Lange, M. Sekoranja, G. Weiss, G. White, G. Jhang, J. Müller, M. Davidsaver, J. León, J. Shannon and T. Ford, "epics-base/epicsCoreJava github repository," 15 December 2023. [Online]. Available: https://github.com/epics-base/epicsCoreJava.

[OSI, "The 3-Clause BSD License," 2023. [Online]. Available: https://opensource.org/license/bsd-3-clause/.

[OSI, "OSI Approved Licences," January 2023. [Online]. Available: https://opensource.org/licenses/.

[F. M. Last Name, "Article Title," *Journal Title,* pp. Pages From - To, Year.

[F. M. Last Name, Book Title, City Name: Publisher Name, Year.

[M. Davidsaver, "ossl.cpp," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/ossl.cpp.

[M. Davidsaver, "ossl.h," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/ossl.h.

[M. Davidsaver, "server.cpp," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/server.cpp.

[M. Davidsaver, "serverchann.cpp," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/serverchan.cpp.

[M. Davidsaver, "serverconn.cpp," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/serverconn.cpp.

[M. Davidsaver, "serverconn.h," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/serverconn.h.

[M. Davidsaver, "config.cpp," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/config.cpp.

[M.Davidsaver, "describe.cpp," 7 September 2023. [Online]. Available: https://github.com/mdavidsaver/pvxs/blob/tls/src/describe.cpp.

[K. Kasemir, "SecureSockets.java," 7 September 2023. [Online]. Available: https://github.com/ControlSystemStudio/phoebus/blob/master/core/pva/src/main/java/org/epics/pva/common/SecureSockets.java.

[M. Barrios, "Stack Overflow can we switch between TCP and TLS on same socket," 12 August 2017. [Online]. Available: https://stackoverflow.com/questions/37763038/is-there-any-way-to-use-sslcontext-with-serversocketchannel.

[A. Medvinsky, M. Hur and Excite, "RFC 2712, Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)," October 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2712.

[M. Jones, J. Bradley and N. Sakimura, "RFC 7515, JSON Web Signature (JWS)," May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7515.

[E. Rescorla and Mozilla, "RFC 8446, TLS 1.3," August 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446.

[J. Linn, "RFC 2743, Generic Security Service Application Program Interface," January 2000. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2743.

[H. Hotz and R. Allbery, "RFC 6717, kx509 Kerberized Certificate Issuance Protocol," August 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6717.

10. Index

Abstract, 2
Application Pillar, 8
Architecture, 20
ASG, 9
authenticate, 19
authorization, 2
AUTHZ, 33
Background, 8
BASE64, 16
Beacon, 22
BKS, 16
CA, 9, 15, 22, 46
Certificate Authority, 9, 15
certificate management, 19
Certificate Management, 11
certificate status, 17, 49
certificates, 19
CISA, 13
Client Certificate, 9
client certificates, 46
client_cert, 26
clientAuth, 17
ClientHello, 48
Control System, 10
CRL, 10, 17
Data at rest, 9
Data in flight, 9
Data in use, 9
digital certificate, 9, 10, 15, 17
digitalSignature, 16, 17
Diskless, 9
Diskless Server, 9, 45
DOD, 13
encapsulation, 17, 18
encrypt, 9, 19
Endpoint, 10
environment variables, 24, 26, 27, 30, 31, 32, 53
EPICS, 2, 10
EPICS “security, 19
EPICS Security, 1, 10, 11
EPICS_PVA_SERVER_PORT, 26, 30
EPICS_PVA_TLS_OPTIONS, 51
EPICS_PVAS_SERVER_PORT, 26, 30
EPICS_PVAS_TLS_OPTIONS, 51
Glossary, 9
HTTPS, 12
Identity Pillar, 8
IOC, 10
IOC Shell, 10
IP Protocol, 10
JWT, 10, 35, 42
KDC, 10
Kerberos, 10, 35, 40
KeychainStore, 16
keyEncipherment, 16, 17
KeyStore, 16
LDAP, 35, 40
List of Figures, 6
Network Pillar, 8
NIST, 13
NIST 800-207, 13
OAuth, 10, 35, 42
OCSP, 10, 11, 17, 48
Online Certificate Status Protocol, 10, 17
openSSL, 52
particle accelerators, 2
PEM, 16
Phoebus, 19
PKCS #12, 11
PKCS#12, 16, 26, 28, 29, 30, 31, 32
ports, 21, 22
private key, 11
Process Variable, 11
Project Goals, 8
protocol stack, 17
public key, 15
PV, 11, 33, 34, 44, 45
PVA, 11
PVA Nameserver, 10, 17
PVAccess, 11
PVAccess gateway, 49
PVAccess Protocol, 19
pvas, 26, 27, 30, 31, 32, 34, 36, 39, 51, 53
PVXS, 19, 49, 52, 53
RFC 6961, 48
Search Response, 34
Secure PVAccess Gateway, 25
serverAuth, 16
SSL, 12
TCP, 12
TCP/IP, 17, 44
TGT, 10
Ticket Granting Tickets, 40
TLS, 12, 19
TLS handshake, 17, 48, 49
token, 10, 35, 42
Transport Layer Security, 12
UAG, 9
vulnerabilities, 8
X.509, 9, 11, 16, 17, 19
Zero Trust, 8, 12, 13