OPC UA Security

FROM ENCRYPTION BASICS TO SECURE EPICS INTEGRATION

Ralph Lange

Background: Cryptography
Encryption 101

- Simplest traditional form:
  Symmetric (shared key) encryption
  - Both peers use the same key
  - Key needs to be shared using a secure path
- Base of modern security:
  Asymmetric (key pairs) encryption
  - Uses complex mathematical *one-way* calculations/algorithms
  - What key A encrypts, *only* key B can decrypt
    What key B encrypts, *only* key A can decrypt

Foundation: Asymmetric Key Pairs

Used for modern security protocols

- **Public key**
  - Shared with everyone
  - Used to *Encrypt* data or *Verify* a signature
- **Private key**
  - Kept secret on the device
  - Used to *Decrypt* data or *Create* a signature.
- Key length and algorithm determine the safety level
  - Be aware of upcoming quantum computing

Workflow: Encryption vs. Signing

Making key pairs useful

Encryption (Confidentiality)

1. Sender uses Receiver's **Public Key** to lock the message.
2. Only the Receiver's **Private Key** can open it.

Signing (Integrity and Authenticity)

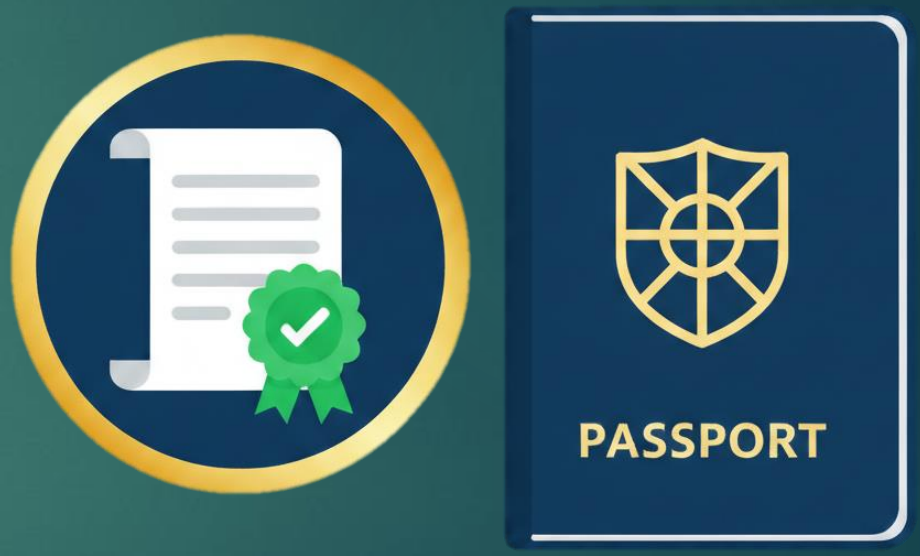
1. Sender hashes the message
2. Sender encrypts it with their own **Private Key**.
3. Receiver uses Sender's **Public Key** to decrypt the hash.
4. If it matches, the identity of the sender and the integrity of the message is proven.

X.509 Certificates

Binding a Public Key to an Identity

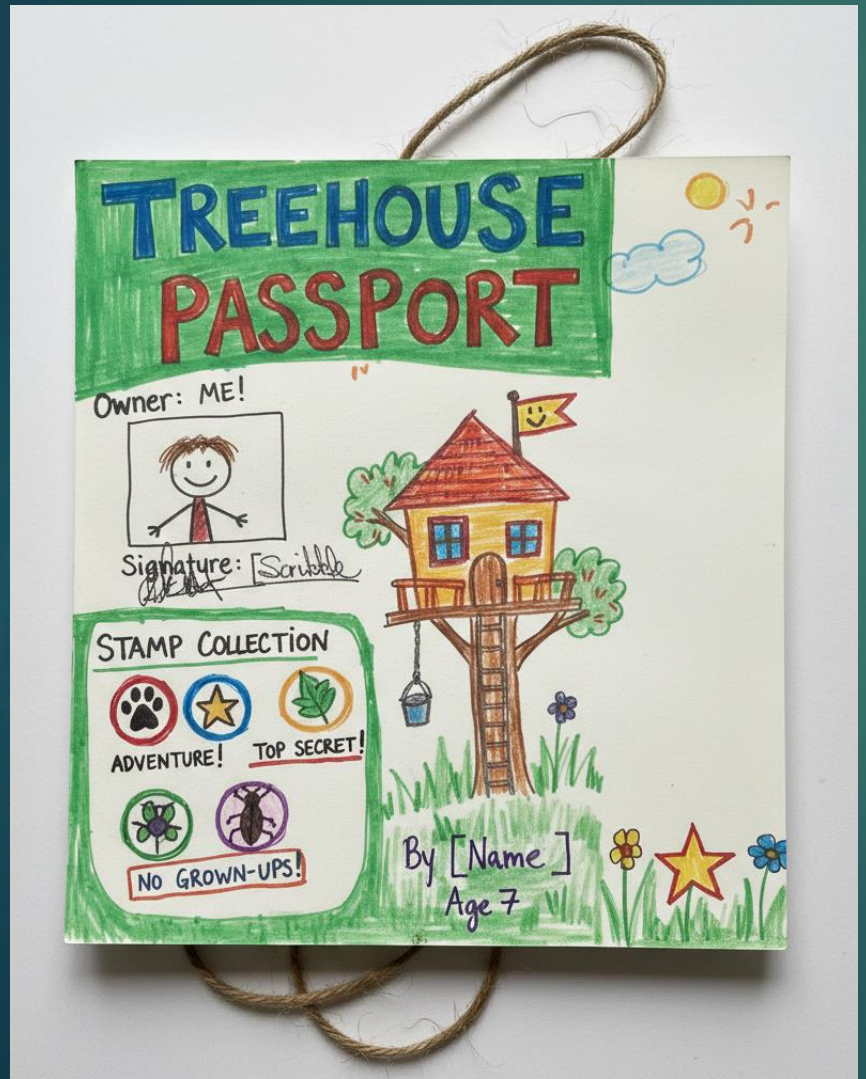
- How do I know the Public Key I just received actually belongs to the server I think it does?
- The Certificate – a "Digital Passport" for your OPC UA Application

  A certificate is a digital document that binds a Public Key
  to an identity (Application Name, URI, Hostname).



Trust Models: Self-Signed Certificate

Kids' Play



- I claim I am who I am
  I signed the document myself
- *Use Case:* Small labs, internal testing, "Trust on First Use"

- Does not scale well: peers need to individually trust every certificate
- Easy to get working; great for development

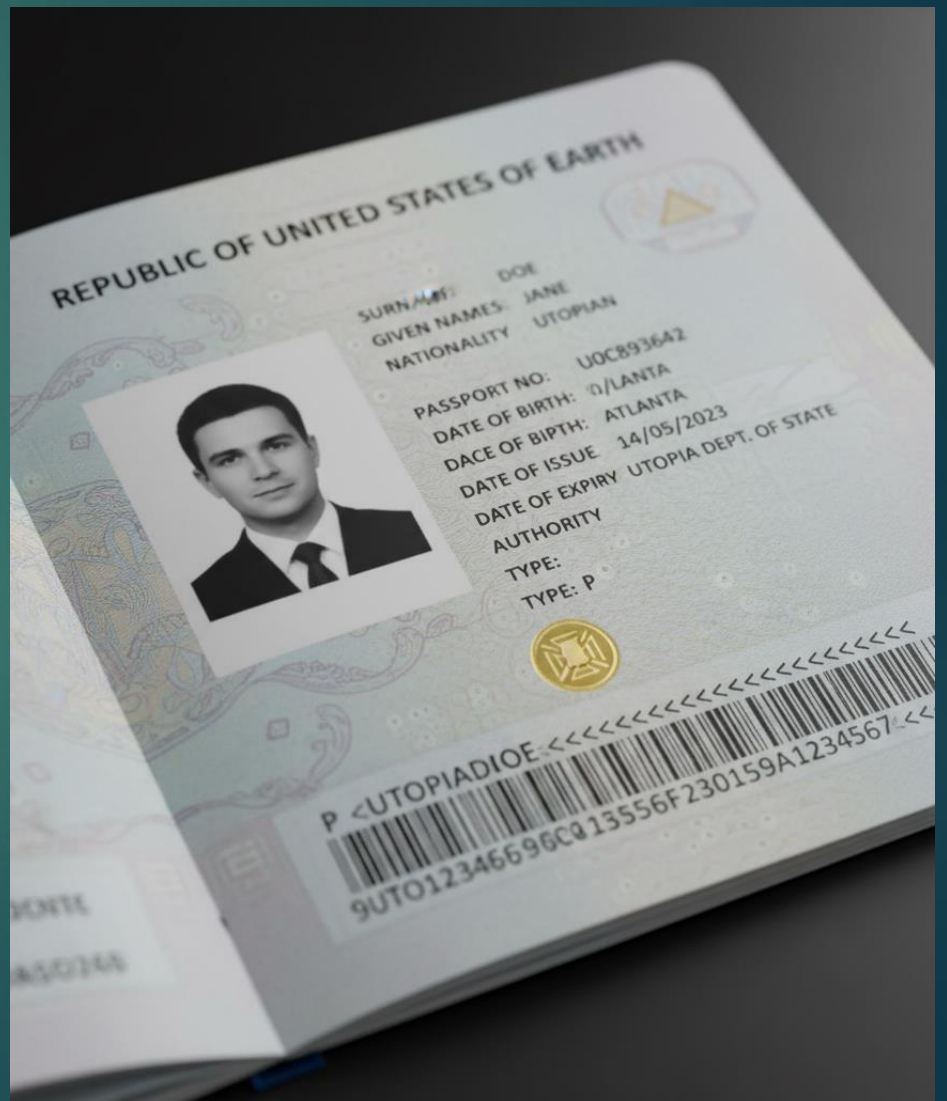
EPICS

Trust Models: CA-Signed Certificate

The Authority

- A trusted third party (Certificate Authority) verified my ID and signed the document for me
- *Use Case:* Production plants, enterprise-wide security

- Peers only need the CA certificate
- Adds considerable complexity and maintenance effort



Fact Mapping: Identification

- **X.509 Cert**
  Certificate Fields: Subject Name, Application URI, Hostname
- **Passport**
  Full Name, Date of Birth, Place of Residence

- Both prove *who* the holder is supposed to be.
  If the Hostname on the cert doesn't match the IP the client connects from, the "Passport" is invalid for that person.

Fact Mapping: The Public Key

- **X.509 Cert**
  Certificate Field: "Public Key Info“ (RSA/ECC)
- **Passport**
  ID Photo, fingerprint

- Photo and fingerprint allow you to recognize/verify the person in front of you.
  The Public Key allows the OPC UA Server to verify signatures and recognize encrypted traffic from the Client.

Fact Mapping: Validation & Trust

- **X.509 Cert**
  Issuer Name and Digital Signature
- **Passport**
  Government Seal and Issuing Authority (e.g., Dept. of State)

- You trust the Passport because you trust the Government that signed it.
  In OPC UA, you trust the Certificate because you have the CA's Certificate in your **Trusted** folder.

Secure OPC UA: The Handshake

1. **Get Endpoints**
   Client asks Server for security options
2. **Open Secure Channel**
   Client and Server exchange certificates
3. **Check Trust**
   Both sides check if the received cert is in their "Trusted" list (or signed by a trusted CA)
4. **Create Session**
   Identity is verified
   Encrypted communication begins

Security Policies in OPC UA

- **None**
  No security – data in cleartext
- **Sign**
  Ensures data isn't tampered with, but still cleartext on the network
- **SignAndEncrypt**
  The gold standard – data is private; source and integrity are verified
- **Algorithms**
  Extensible list (to keep up with crypto developments), e.g.
  Basic256Sha256 (common)
  Aes128Sha256Rsa15 (older)
  …

CA Chains of Trust

- For easier certificate management, CA signatures can be chained:

RootCA self-signed root CA certificate – kept in the safe

signs

CA1 **CA2** **CA3** e.g. different locations

signs *signs* *signs*

CA11 **CA12** **CA21** **CA22** **CA31**

sign

100s of client and server certificates

- For verification, peers need the complete chain of CA certificates from the presented server or client certificate to the root CA
- Certificate Revocation Lists (CRLs) are signed by a CA and distributed to invalidate specific signatures

The PKI File Store

- PKI = Public Key Infrastructure
  tools and procedures to manage public keys for encryption
- PKI File Store for OPC UA
  Standard directory structure for keeping public keys

trusted/certs	trusted peer and CA certificates of the other side
trusted/crl	revocation lists for the CAs in **trusted/certs**
issuers/certs	intermediate CA certificates for the own certificate
issuers/crl	revocation lists for the intermediate CAs

- Certificates and CRLs must be in DER format
  Keys must be in PEM format

Top Reasons Why Connections Fail

*Be prepared for a ***lot*** of frustration*

- **Untrusted Certificate:** The Server's cert is not in the Client's trusted/certs folder (or vice versa)
- **Hostname Mismatch:** Connecting via IP, but the certificate only lists the DNS hostname
- **URN Mismatch:** The URN that the IOC uses does not exactly match the one in the certificate
- **Clock Desync (NTP):** If the Linux system time is 2010, but the cert was created in 2024, the cert is "not yet valid"
- **Expired Certificate:** The "Valid Until" date has passed
- **CRL (Revocation List):** The system is trying to check if the cert was revoked but doesn't have a valid Revocation List

Fasten Your Seat Belts

- We will now extend the OPC UA connection from the basic training to use Secure OPC UA between the IOC and the UA Demo server
- You can follow along in the Training VM or watch and do it yourself later

- But first:

Any questions up to this point ??